

Apunte de Final para Base de Datos

Departamento de Computación – UBA

Contenido

Modelización	2
Consultas	4
Álgebra Relacional (AR)	4
Cálculo Relacional de Tuplas (CRT).....	4
Structured Query Language (SQL)	5
Normalización.....	6
Optimización	9
Organización de Archivos	9
Optimizador de Consultas (Query Optimizer).....	10
Transacciones	11
Locking.....	13
Recuperabilidad.....	13
Logging	14
Undo	14
Redo	14
Undo/Redo	15
Mapeo Objeto Relacional.....	16
XML.....	17
Data Warehousing.....	19
Modelado de Datos	23
Data Mining	26
No SQL Databases	33
DBA	35
Bases de Datos Paralelas y Distribuidas	36
Bases de Datos Paralelas – Detalle.....	37
Finales Resueltos	43
11/04/2012.....	43
Fines del 2012.....	45
30/07/2013.....	46

Modelización

Base de Datos: Una base de datos o banco de datos es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.

Modelo de Datos: Un modelo de datos es un lenguaje orientado a hablar de una Base de Datos.

Típicamente un modelo de datos permite describir:

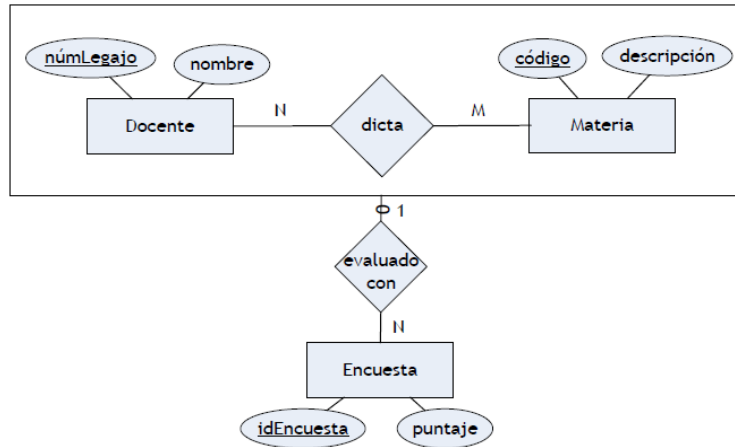
- Las estructuras de datos de la base: El tipo de los datos que hay en la base y la forma en que se relacionan.
- Las restricciones de integridad: Un conjunto de condiciones que deben cumplir los datos para reflejar correctamente la realidad deseada.
- Operaciones de manipulación de los datos: típicamente, operaciones de agregado, borrado, modificación y recuperación de los datos de la base.

Otro enfoque es pensar que un modelo de datos permite describir los elementos de la realidad que intervienen en un problema dado y la forma en que se relacionan esos elementos entre sí.

No hay que perder de vista que una Base de Datos siempre está orientada a resolver un problema determinado, por lo que los dos enfoques propuestos son necesarios en cualquier desarrollo de software.

- Cómo se diseña una base de datos:
 - o Análisis de Requerimientos: Que espera el usuario de la BD
 - o Diseño Conceptual: Se arma un modelo de alto nivel (generalmente MER). Debe respetar los requerimientos de la situación real del usuario. Se debe evitar la redundancia
 - o Diseño Lógico: Modelo más cercano a la implementación, si el motor de la BD es relacional lo diseñamos directamente en el modelo relacional
 - o Refinamiento: Se normalizan/des-normalizan las tablas
- Tipos de modelado:
 - o Modelo de Entidad-Relación: Representa a las entidades, sus relaciones y atributos
 - Ventajas: Alto Nivel, y por lo tanto cercano a como los usuarios perciben los datos
 - Componentes:
 - Entidades: Representan conceptos u objetos de interés, se describen con atributos y poseen una clave que lo identifica
 - o Jerarquía: Cuando varias entidades comparten características se pueden agrupar estas en otra entidad más grande (superentidad). Las más chicas (subentidades) heredan la clave del padre. Estas tienen 2 características:
 - Cobertura: Total cuando cada elemento tiene que estar en una subentidad, o parcial cuando no necesariamente es así
 - Solapamiento: Disjunto cuando un elemento puede estar en hasta una subentidad o con solapamiento cuando un elemento puede estar en más de una subentidad
 - o Entidades débiles: Necesitan a otra para identificarse
 - El atributo identificador de la entidad es parte de la otra entidad. Dicha participación siempre es total

- Relaciones: Poseen Grado (cantidad de entidades (unaria, binaria, ternaria)), Cardinalidad (cantidad de elementos de la(s) otra(s) entidad(es)), y Participación (obligatoria (total) o no (parcial))
 - Pueden tener atributos, los cuales registran información adicional
 - Agregación: En la relación ternaria siempre deben estar los 3 elementos simultáneamente, si quisiéramos que uno no siempre esté presente utilizaríamos este patrón



- Modelo Relacional: Se basa en relaciones (tablas) con atributos (columnas) y tuplas (registros)
 - Ventajas: Más cercano a la implementación
 - Conceptos importantes:
 - Clave: Conjunto de atributos que definen unívocamente a las tuplas
 - Puede haber más de una clave, solo una será la primaria, el resto serán “candidatas”
 - Se puede hacer referencia a la clave de otra tabla, las externas son llamadas foreign key
 - Transformaciones destacadas desde el modelo entidad-relación:
 - Jerarquía disjunta: el padre lleva el atributo “tipo” que indica en que hijo está un elemento
 - Jerarquía con solapamiento: se generan tablas aunque no tengan atributos

Consultas

Álgebra Relacional (AR)

- Definición: Lenguaje de consultas relacionado al modelo relacional (MR)
- Todas las consultas tienen como inputs/outputs las instancias de una relación
- Ninguna instancia de relación tiene repetidos (son conjuntos)
- Cada consulta describe paso a paso como se llega a calcular el resultado
- Operadores:
 - o Básicos: Proyección, Selección
 - o Conjuntos: Unión, Intersección, Resta, Producto Cartesiano
 - o Joins: Con condición, EquiJoin (igualdad), Natural Join (atributos del mismo nombre)
 - o Avanzados: Renombre, División (elementos que están relacionados con todos los elementos de una relación)

- Ejemplo de división:

R		S		R % S
a1 b1		b1		
a1 b2		b2		
a1 b3	%	b3	=	a1
a2 b2				
a3 b3				

- Propiedades (utilizadas en la optimización):
 - o Cascada de σ : $\sigma_{C1 \text{ and } C2 \dots \text{and } Cn} (R) \equiv \sigma_{C1} (\sigma_{C2} (\dots \sigma_{Cn} (R) \dots))$
 - o Conmutatividad de σ : $\sigma_{C1} (\sigma_{C2} (R)) \equiv \sigma_{C2} (\sigma_{C1} (R))$
 - o Cascada de π : $\pi_{list1} (\pi_{list2} (R)) \equiv \pi_{list1 \cap list2} (R)$
 - o Conmutatividad de σ con respecto a π : $\pi_{A1, \dots, An} (\sigma_C (R)) \equiv \sigma_C (\pi_{A1, \dots, An} (R))$ (Si C ref. solamente a atr. dentro de A1...An)
 - o Conmutatividad del Producto Cartesiano (o Junta): $R \times S \equiv S \times R$
 - o Conmutatividad del σ con respecto al Producto Cartesiano: $\sigma_C (R \times S) \equiv (\sigma_{Cr}(R)) \times (\sigma_{Cs}(S))$ donde $C = Cr \cup Cs$
 - o Conmutatividad de π con respecto al Producto Cartesiano: $\pi_L (R \times S) \equiv (\pi_{L1}(R)) \times (\pi_{L2}(S))$ donde $L = L1 \cup L2$
 - o Conmutatividad de operaciones de conjuntos (\cup e \cap): $R \theta S \equiv S \theta R$ (Si $\theta = \{\cup \text{ e } \cap\}$)
 - o Asociatividad del Producto Cartesiano, Junta, \cup e \cap : $(R \theta S) \theta T \equiv R \theta (S \theta T)$ (Si $\theta = \{x, |X|, \cup \text{ e } \cap\}$)

Cálculo Relacional de Tuplas (CRT)

- Definición: Lenguaje de consultas relacionado al modelo relacional (MR)
- Es de tipo declarativo, es decir, se dice lo que se tiene que hacer y no como
- Reciben como inputs/outputs instancias de relación
- El esquema de la consulta es $\{t / F(t)\}$ donde t es libre y F es la fórmula que la describe (en 1er orden)
- Ejemplos:
 - o Listar los nombres de los actores mayores de 30 años que participan en la serie "Friends"
 $\{t / \exists a, s, p (a \in \mathbf{Actor} \wedge a.edad > 30 \wedge s \in \mathbf{Serie} \wedge s.nombreSerie = "Friends" \wedge p \in \mathbf{Participa_En} \wedge p.idActor = a.idActor \wedge p.idSerie = s.idSerie \wedge t.nombreActor = a.nombreActor)\}$
 - o Listar los nombres de los canales que transmiten todas las series de comedia
 $\{t / \exists c (c \in \mathbf{Canal} \wedge t.nombreCanal = c.nombreCanal \wedge \forall s (esSerieDeComedia(s) \Rightarrow \exists tra (tra \in \mathbf{Transmite} \wedge tra.idCanal = c.idCanal \wedge tra.idSerie = s.idSerie)))\}$
Definimos el predicado *esSerieDeComedia* como:
 $esSerieDeComedia(s) = s \in \mathbf{Serie} \wedge \exists g (g \in \mathbf{Género} \wedge g.idGénero = s.idGénero \wedge g.nombreGénero = "Comedia")$
 - o Listar los ids de los actores que participaron en al menos 2 series
 $\{t / \exists a (a \in \mathbf{Actor} \wedge t.idActor = a.idActor \wedge participóEnAlMenos2Series(a))\}$
 $participóEnAlMenos2Series(a) = \exists p, p' (p \in \mathbf{Participa_En} \wedge p' \in \mathbf{Participa_En} \wedge p \neq p' \wedge p.idActor = a.idActor \wedge p'.idActor = a.idActor)$

Structured Query Language (SQL)

Definición: El lenguaje más usado para las bases de datos, es declarativo y de alto nivel

Sentencias:

- Data Definition Language (DDL): Permiten modificar la estructura
- Data Manipulation Language (DML): Permiten manipular los datos
- Se proveen sentencias para definir permisos, manejar transacciones y otros

Estructura básica:

- SELECT /* columnas/expresiones a ser retornadas */
- FROM /* relaciones entre tablas */
- [WHERE /* condic sobre la filas a ser retornadas */]
- [GROUP BY /* atributos de agrupamiento */]
- [HAVING /*cond sobre los grupos */]
- [ORDER BY /*orden en que se retornan las filas*/]

Operadores de conjuntos:

- UNION/UNION ALL: Retorna filas pertenecientes a varias consultas sin/con repetidos respectivamente
- INTERSECT: Retorna filas comunes
- MINUS: Retorna las filas de la 1er consulta que no estén presentes en la 2da
- DISTINCT: Retorna las filas distintas
- LEFT/LEFT OUTER JOIN: Retorna las filas del JOIN normal + aquellas en las cuales el valor de la columna que participa en el join tiene el valor nulo (del lado izquierdo)

Funciones agregadas para el Group By: COUNT, SUM, MAX, MIN, AVG

Consultas anidadas: Consultas dentro de consultas, para acceder a ellos se utilizan operadores de single-row si devuelven un resultado (como =, <, >) u operadores de multiple-row si devuelven muchos resultados (como ALL, ANY, IN, EXISTS)

Stored Procedure: Es una porción de código, que se almacena en el catálogo de la base de datos y se puede invocar mediante una sentencia SQL. Encapsulan reglas de negocio fuertemente relacionadas con los datos de la BD y sin interacción con el usuario. Tienen un nombre, reciben parámetros y pueden devolver resultados. Permiten reutilizar código

Cursor: Es un puntero que apunta a una fila que pertenece al conjunto de registros (result set) resultantes de un SELECT

Trigger: Código almacenado en la DB que se ejecuta ante ciertos eventos (After/instead of Insert/Delete/Update)

Consulta clásica del AR en SQL: División:

```
SELECT S.sname
FROM sailors S
WHERE NOT EXISTS
  (SELECT B.bid
   FROM Boats B
   WHERE NOT EXISTS
     (SELECT R.bid
      FROM Reserves R
      WHERE R.bid=B.bid
            AND R.sid=S.sid))
```

Obtener los marinos S tales que...

No existe ningún bote B ...

Sin una reserva a nombre del marino S

Índices: Estructura de acceso físico a datos que sirve para acceder más rápidamente a las filas de las tablas, de manera independiente

Vistas: Son relaciones donde solo almacenamos su definición y no su conjunto de filas

Normalización

Definición: El proceso de normalización de bases de datos consiste en aplicar una serie de reglas a las relaciones obtenidas **tras el paso del modelo entidad-relación al modelo relacional**. Lo que se busca es: evitar redundancia, evitar problemas de actualización, y proteger la integridad de los datos

- Introducción:

- Vale $X \rightarrow Y$ en R si para toda r se verifica que: $t_1(X) = t_2(X) \rightarrow t_1(Y) = t_2(Y)$. X determina funcionalmente a Y
- Reglas de Inferencia
 - Axiomas de Armstrong
 - Reflexividad: X está contenido en Y $\rightarrow X \rightarrow Y$
 - Aumento: Para cualquier W, $X \rightarrow Y \rightarrow XW \rightarrow YW$
 - Transitividad: Si $X \rightarrow Y$, $Y \rightarrow Z \rightarrow X \rightarrow Z$
 - Adicionales
 - Unión: $X \rightarrow Y$, $X \rightarrow Z \rightarrow X \rightarrow YZ$
 - Pseudotransitividad: Para cualquier W, Si $X \rightarrow Y$ e $YW \rightarrow Z \rightarrow XW \rightarrow Z$
 - Descomposición: Si $X \rightarrow YZ \rightarrow X \rightarrow Y$, $X \rightarrow Z$
- Clausura de DF: Conjunto de DF que pueden inferirse a partir de F aplicando reglas de inferencia

$$F^+ = \{X \twoheadrightarrow Y / F \models X \twoheadrightarrow Y\}$$

- Clausura de Conjunto de Atributos: Conjunto de atributos tal que $X \rightarrow A$

$$X^+ = \{A \in R / F \models X \twoheadrightarrow A\}$$

- Forma de computarlos:
 - X_0 es X
 - $X_{i+1} = X_i \cup \{ \text{conjunto A} / \text{hay DF } Y \rightarrow Z \text{ en F, A está en Z e Y está en } X_i \}$
 - Para cuando $X_{i+1} = X_i$
- Propiedades:
 - Dados F y $X \rightarrow Y$, luego F fuerza $X \rightarrow Y \Leftrightarrow Y$ está en X^+
 - Si $X^+ = R$, entonces X es **superclave** de R
- **Clave Candidata:** Es una superclave de la relación tal que ningún subconjunto propio es superclave
- **Clave Primaria:** Es la clave candidata cuyos valores se utilizarán para identificar las tuplas de la relación
- **Clave Alternativa:** Son las claves candidatas no elegidas como primaria
- Algoritmo para encontrar todas las claves de R
 - Computamos primero los atributos que no están en ningún lado derecho, llamémoslos X
 - Si $X^+ = R$, X es la única clave candidata
 - Sino hay que probar todos los casos, es decir: comenzamos con $X \cup A$ para cada A
 - Si $XA^+ = R$, XA es CC y todos los que incluyen a XA serán superclave
 - Si $XA^+ \neq R$ agregamos un atributo más a XA, pónelo B y computamos XAB^+ hasta que sea igual a R
- Equivalencia de conjuntos de DF: F Equivale a G Si y Solo Si $F^+ = G^+$, o bien F fuerza G y G fuerza F
- Cubrimiento Minimal: Dado F busquemos F_m tal que F_m equivalga a F. F_m tiene:
 - Todo lado derecho tiene un único atributo
 - Todo lado izquierdo es reducido (no tiene atributos redundantes)
 - No contiene DF redundantes (para verificar se saca la DF $(X \rightarrow Y)$ y se busca saber si X^+ en F contiene a Y, si lo contiene era porque $X \rightarrow Y$ era redundante)
- Atributo Primo: Será aquel que es miembro de Alguna Clave Candidata

- Formas Normales
 - o 1era: Si R tiene todos sus atributos atómicos, en el modelo relacional todo esquema está en 1FN por def.
 - o 2da: Si No hay atributos en R que dependen parcialmente de una clave
 - o 3ra: R está en 3FN si para toda DF no trivial $X \rightarrow A$ sobre R: **X es superclave de R o A es primo**
 - Es decir: no pueden haber lados izq. sin ser superclaves y al mismo tiempo lado der. no primo
 - Ejemplo: $A \rightarrow B$, A superclave? Sí, OK; $A \rightarrow C$, A superclave? Sí, OK; $B \rightarrow C$, B superclave? No. C primo? No. Se está violando la 3FN
 - o FNBC (Forma Normal Boyce Code): **Todos los lados izquierdos son superclaves**
 - Algoritmo para obtener una descomposición en 3FN, SPI, y SPDF:
 - o 1) Obtener Cobertura Minimal de F
 - o 2) Cada DF será un esquema (si hay mismo lado izq. se pueden juntar)
 - o 3) Si ningún esquema contiene alguna clave, se agrega un esquema con la clave
 - o 4) Si un esquema contiene a otro, a este último lo removemos
 - Algoritmo para obtener una descomposición en FNBC, SPI:
 - o 1) Se aplican descomposiciones binarias mientras que exista $X \rightarrow Y$ que viole la FNBC
 - Es decir: R se descompone en $R_1=R-Y$, $R_2=XY$ y así sucesivamente
 - o 2) Luego, cada hoja del árbol será un esquema
 - Cómo corroborar si una descomposición mantiene las dependencias funcionales (DF)
 - o Significa que todas las dependencias originales se pueden obtener a partir de las dependencias que se proyectaron sobre los subesquemas. No siempre es posible cumplirla
 - o Proyección de un conjunto de dependencias funcionales: Sea F un conjunto de DF, decimos que esta proyección sobre un conjunto de atributos Z ($Plz(F)$) es el conjunto de dependencias funcionales $X \rightarrow Y$ en F+ tal que XY estén contenidas en Z (es decir, si agregamos atributos agregamos sus dependencias)
 - o Luego, decimos que p es una descomposición **SPDF** (sin pérdida de DF) si:
$$F^+ = (\bigcup_{i=1}^k \Pi_{R_i}(F))^+$$
 - o Algoritmo para testear pérdidas de DF:
 - Dados $R=\{A_1, \dots, A_n\}$, F, y $p=\{R_1, \dots, R_k\}$
- Para toda dependencia funcional $X \rightarrow Y \in F$:
- Verificar que se preserva $X \rightarrow Y$:
- $Z = X$
- while Z cambia
- for $i = 1$ to k do
- $Z = Z \cup ((Z \cap R_i)^+ \cap R_i)$ (la clausura de $(Z \cap R_i)$ es con r respecto a F)
- Si $Y \not\subseteq Z$ retornar No.
- Retornar Sí.
- Ejemplo de utilización:
 - Sea $R=\{A,B,C,D,E\}$, $F=\{AB \rightarrow C, A \rightarrow D, D \rightarrow E, E \rightarrow C\}$
 - Decidir si $p = \{(A,D), (D,E), (E,C,B)\}$ es SPDF
 - Viendo el algoritmo, tomamos $Z = AB$, y vemos si llegamos a C
 - o $Z = Z \cup ((Z \cap R_1)^+ \cap R_1) = \{AB\} \cup ((\{AB\} \cap \{A,D\})^+ \cap \{A,D\}) = \{AB\} \cup (\{A\}^+ \cap \{A,D\}) = \{AB\} \cup \{A,D\} = \{ABD\}$ (no está C)
 - o $Z = Z \cup ((Z \cap R_2)^+ \cap R_2) = \{ABD\} \cup ((\{ABD\} \cap \{D,E\})^+ \cap \{D,E\}) = \{ABD\} \cup (\{D\}^+ \cap \{D,E\}) = \{ABD\} \cup \{D,E\} = \{ABDE\}$ (no está C)
 - o $Z = Z \cup ((Z \cap R_3)^+ \cap R_3) = \{ABDE\} \cup ((\{ABDE\} \cap \{E,C,B\})^+ \cap \{E,C,B\}) = \{ABDE\} \cup (\{EB\}^+ \cap \{E,C,B\}) = \{ABDE\} \cup \{E,C,B\} = \{ABDEC\}$ (está C)
 - El resto de las dependencias se conserva trivialmente, p conserva la dependencia

- Cómo corroborar que una descomposición es sin pérdida de información (SPI)

- Decimos que una desc. es SPI si para cada instancia r de R que satisface F se verifica:

$r = \bowtie_{i=1}^k \prod_{R_i}(r)$, si se puede recuperar la rel. original a partir de las desc. mediante la junta natural.

Siempre es posible cumplirla

- Teorema: Descomposición binaria

- La descomposición p de R , $p=\{R_1, R_2\}$ es SPI respecto a un conjunto de dependencias funcionales

Si y Solo Si F^+ contiene la DF $R_1 \cap R_2 \rightarrow (R_1 - R_2)$ o bien F^+ contiene la DF $R_1 \cap R_2 \rightarrow (R_2 - R_1)$

- Descomposición de 2 esquemas: se comprueba mediante esta descomposición binaria, en cambio si es más se utiliza el **Algoritmo de Tableau**

- Algoritmo de Tableau:

Dados $R = \{A_1, A_2, \dots, A_n\}$, F y $\rho = \{R_1, R_2, \dots, R_k\}$

Construir una tabla T , donde las columnas son los atributos de R y las filas los subesquemas de ρ .

Completar la tabla, asignando a T_{ij} el símbolo distinguido a_j si $A_j \in R_i$ y el no distinguido b_{ij} sino.

Para cada dependencia funcional $X \rightarrow Y$

Para cada par de filas f_k, f_r , con $r > k$, tales que:

$$f_k[X] = f_r[X]$$

$$f_k[Y] \neq f_r[Y]$$

Si $f_k[Y] = a_y$ o bien $f_r[Y] = a_y$ poner a_y en ambas.

Sino poner $f_k[Y]$ en ambas.

Si hay una fila con todos los símbolos distinguidos, retornar Sí.

Mientras T se modifique repetir el 3er paso.

Retornar No.



- Ejemplo de utilización:

- Sea $R = \{A, B, C, D, E\}$, y $F = \{A \rightarrow B, D \rightarrow C\}$
- Decidir si la descomposición $p = \{(A, B, C), (C, D, E), (A, D, E)\}$ es SPI.
- 1) Construir una tabla T , donde las col. son los atributos de R y las filas los subesquemas de p .
- 2) Completar la tabla, asignando a T_{ij} el valor A_j si A_j pertenece a R_i y A_j sino.

	A	B	C	D	E
ABC	a_1	a_2	a_3	b_{14}	b_{15}
CDE	b_{21}	b_{22}	a_3	a_4	a_5
ADE	a_1	b_{32}	b_{33}	a_4	a_5

- 3) Para cada dependencia funcional $X \rightarrow Y$, para cada par de las f_k, f_r , con $r > k$, tales que: $f_k[X] = f_r[X]$, $f_k[Y] \neq f_r[Y]$; Si $f_k[Y] = a_y$ o bien $f_r[Y] = a_y$, poner a_y en ambas. Sino poner $f_k[Y]$

$A \rightarrow B$

	A	B	C	D	E
ABC	a_1	a_2	a_3	b_{14}	b_{15}
CDE	b_{21}	b_{22}	a_3	a_4	a_5
ADE	a_1	a_2	b_{33}	a_4	a_5

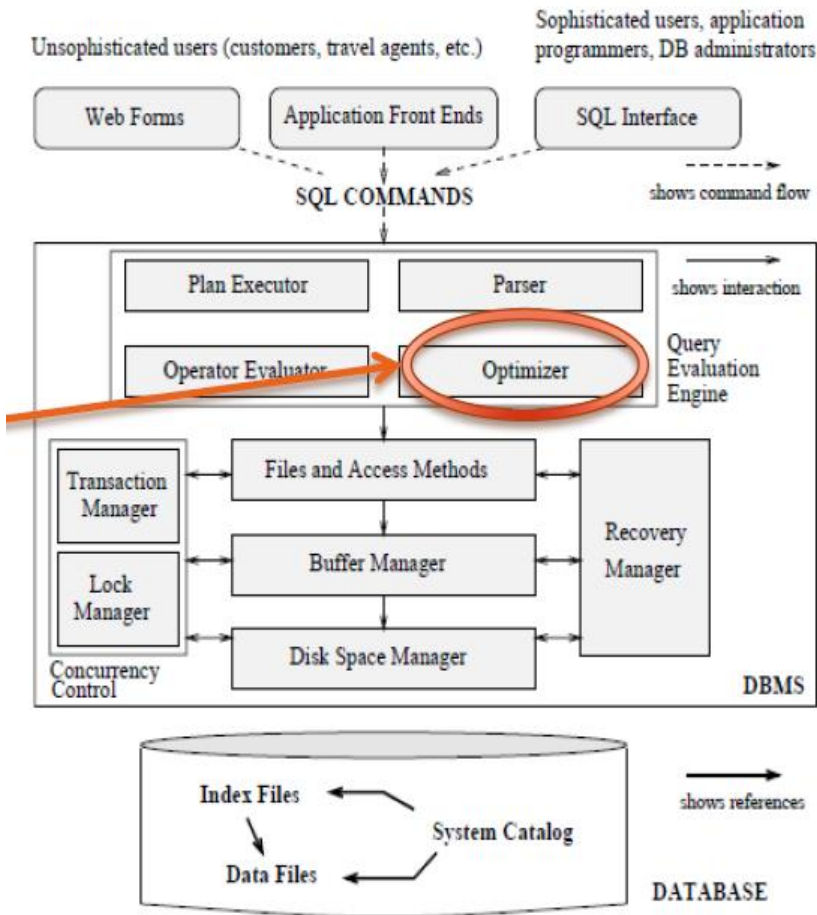
- 4) Si hay una fila con todos los símbolos distinguidos, retornar Si.
- 5) Mientras T se modifique repetir el 3er paso, si no se llega a nada retornar No

	A	B	C	D	E
ABC	a_1	a_2	a_3	b_{14}	b_{15}
CDE	b_{21}	b_{22}	a_3	a_4	a_5
ADE	a_1	a_2	a_4	a_4	a_5

- Luego p era SPI

Optimización

Arquitectura de una base



Disk Manager:

- Capa de más bajo nivel de la BD que maneja el espacio en disco, oculta los detalles sobre como se almacenan las tablas en hardware
- Provee Abstracción haciendo que las tablas se vean como colección de páginas

Buffer Manager:

- Responsable de traer las páginas de disco a memoria. Administra dicho espacio, dividiendo a las páginas en bloques de igual tamaño
- Provee Abstracción haciendo que no importe si las páginas están en disco o memoria

Query Evaluation Engine:

- Recibe comandos SQL, el **Parser** los parsea, y se los pasa al **query optimizer** quien busca un plan de ejecución eficiente
- Finalmente el plan lo ejecuta el **query evaluator**

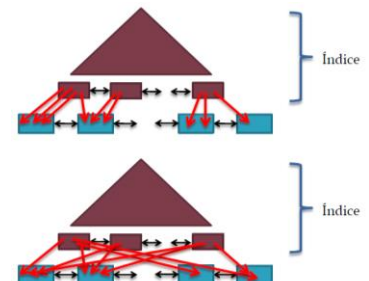
Transaction Manager: Controla las transacciones

Lock Manager: Controla los bloqueos

Recovery Manager: Mantiene un log sobre los cambios en la base, y si esta se cae se encarga de dejarla consistente

Organización de Archivos

- Las BD se almacenan como colección de archivos
- Cada archivo contiene registros del mismo tipo y se dividen en bloque de igual tamaño
- Tipos
 - o **Heap File:** No siguen orden, en los inserts se busca un espacio libre o se agrega al final del archivo.
 - Las operaciones de búsqueda suelen requerir una búsqueda lineal en todo el archivo
 - o **Sorted File:** Se ordenan a partir de una clave de búsqueda, y en los inserts se mantiene el orden
 - Las operaciones de búsqueda sobre esa clave son en $\log(\#cantRegistros)$, el resto lineal
- **Índices:** Estructuras adicionales para acelerar ciertas búsquedas a costo de la complejización del ABM. Tipos:
 - o **B+: Árboles Balanceados**
 - **Clustered:** Archivo e índice contienen mismo orden
 - **Unclustered:** Archivo e índice se ordenan distinto
 - o **Hash:** Se arma tabla de Hash (generalmente estática) donde almacenar las claves de búsqueda. En cada bucket puede haber cantidad variable de bloques (podría haber aglomeramiento según una misma clave)



Optimizador de Consultas (Query Optimizer)

- Objetivo: Dada una consulta, encontrar un plan de ejecución eficiente
- Un plan de ejecución define cómo se resolverá una consulta dada, indicando paso a paso los algoritmos y estructuras que se usarán para resolverla
- Eficiente y no mejor plan porque hallar el mejor es tan costoso que termina tardando más que la misma query...
- Construcción del Query Plan
 - o A) SQL -> AR -> Árbol Canónico
 - o B) Modificaciones algebraicas sobre el árbol
 - Las técnicas utilizan heurísticas basadas en propiedades algebraicas (es el enfoque clásico de reglas):
 - Descomponer las selecciones conjuntivas en una secuencia de selecciones simples formadas por cada uno de los términos de la selección original
 - Permutar las relaciones de las hojas para evitar productos cartesianos
 - Bajar las selecciones que son condiciones de junta para poder cambiar productos cartesianos por natural joins
 - Llevar las selecciones lo más cercano posible a las hojas del árbol, de manera de lograr la ejecución temprana reduciendo así el número de tuplas que se propagan hacia niveles superiores
 - Descomponer las listas de atributos de las proyecciones y llevarlas lo más cercano posible a las hojas del árbol, creando nuevas proyecciones cuando sea posible de manera de no propagar hacia niveles superiores atributos innecesarios. De esta manera se logra una reducción temprana del tamaño de las tuplas, y se reduce la cantidad de bloques necesaria para almacenamiento intermedio
 - Tener en cuenta los índices interesantes al momento de generar los planes de ejecución. En muchas ocasiones puede ser importante no “bajar” al máximo las proyecciones y/o selecciones sobre una relación, de manera de poder aprovechar el índice de la relación en alguna operación de un nivel superior, y aplicar recién el filtro y/o la proyección luego de esta operación.
 - Utilizar el pipeline entre las operaciones siempre que sea posible, de manera de evitar los costos adicionales a causa de dar persistencia a disco a resultados intermedios en forma innecesaria.
 - Técnicas Físicas (es un modelo basado en costos que utiliza mayor inteligencia en sus planes):
 - Consisten en seleccionar implementaciones para los operadores basándose en como están organizados los archivos y las estructuras adicionales que existen
 - Utilizan una estructura de la BD llamada **Catálogo**
 - o Mantiene información estadística acerca de los datos de las diferentes tablas.
Ejemplos: Número de tuplas de la relación R; Tamaño de una tupla de R; Cardinalidad del dominio del atributo A; Número de valores distintos del atributo A; Valor máximo del atributo A; Valor mínimo del atributo A
 - o Se actualiza periódicamente y no están siempre sincronizadas con los datos reales
 - o Permite estimar la selectividad de los diferentes operadores
 - o Todas las tablas e índices que creemos, entre otros, se registran allí
 - o Las consultas al catálogo pueden realizarse con las mismas sentencias que se consulta cualquier base de datos o tabla. Sin embargo las actualizaciones al catálogo (INSERT, DELETE, UPDATE) no son posibles.
 - o C) Implementación de cada operador (ejemplo: definir como se realizará un join)

- Comparación de planes: Se define un Modelo de Costos donde:
 - o El costo se expresa en accesos al disco (lecturas y escrituras), y es estimativo
- Pasaje de resultados entre nodos:
 - o Materialización: Los resultados intermedios se guardan a disco, y el siguiente nodo deberá levantarlos
 - o Pipeline: Las tuplas se van pasando al nodo superior mientras se continúa ejecutando la operación
- JOINS:
 - o Block Nested Loops Join (BNLJ): Se llenan Total-2 bloques de memoria con una relación mientras se comparan todos los valores de la otra con los bloques restantes, cuando se terminan los bloques restantes se vuelven a llenar esos Total-2 bloques con los siguientes de la 1er relación se repite todo hasta haber recorrido las 2 rel.
 - o Index Nested Loops Join (INLJ): Se recorre una relación y se busca por índice en la 2da
 - o Sort Merge Join (SMJ): Se ordenan ambas relaciones y luego se busca ordenadamente
 - o Costos

Tipo de archivo / índice	Costo de exploración completa	Costo de búsqueda por igualdad ($A = k$)	Costo de búsqueda por rango ($k_1 \leq A \leq k_2$)
Heap file	B_R	B_R	B_R
Sorted file	B_R	$\log_2(B_R) + [T' / FB_R]$	$\log_2(B_R) + [T' / FB_R]$
Índice B+ clustered sobre A	B_R	$X + [T' / FB_R]$	$X + [T' / FB_R]$
Índice B+ unclustered sobre A	B_R	$X + T'$	$X + [T' / FB_R] + T'$
Índice hash estático sobre A	B_R	$MB \times B_i + T'$	B_R

Transacciones

- **Definición:**
 - o Una transacción T es una ejecución de un programa P que accede a la BD. Un mismo programa P puede ejecutarse varias veces. Cada ejecución de P es una transacción T_i .
 - o Una transacción es una sucesión de acciones (u operaciones)
 - Una acción es un paso atómico
 - **leer** un ítem X: $r_i[X]$; **escribir** un ítem X: $w_i[X]$; **abort** de T_i : ai ; **commit** de T_i : ci
 - o Las T_i se ejecutan de manera **concurrente** generando interferencia
 - o Cuando las ejecuciones generan fallas, causan lo que llamamos **problema de recuperación**
- Problemas clásicos
 - o Lost Update: 2 transacciones leen el mismo valor anterior del ítem y luego lo actualizan en forma sucesiva
 - o Dirty Read: ocurre cuando una transacción T_2 lee un valor de un ítem dejado por otra transacción T_1 que no hizo commit antes de que T_2 leyera el ítem. Si T_1 aborta, T_2 quedó con un valor sucio, y podría provocar aborts en cascada (T_1 aborta y obliga a abortar a T_2)
 - o Non-Repeatable Read: El non-repeatable read o lectura no repetible ocurre cuando una transacción T_1 lee un ítem, otra transacción T_2 lo modifica y luego T_1 vuelve a leer ese ítem y ve que ahora tiene otro valor
 - o Phantom Read: Ocurre cuando una transacción T_1 ejecuta un query y lee un conjunto de ítems (generalmente tuplas), otra transacción T_2 inserta nuevos ítems y luego T_1 vuelve a ejecutar el mismo query y ahora el conjunto de ítems ha cambiado. Caso particular del Non-Repeatable Read

- **Propiedades ACID:** (La idea es que dada una BD en un estado consistente, luego de ejecutarse las transacciones la BD quede también en un estado consistente. Una forma de garantizar esto último es que las transacciones cumplan con las propiedades ACID)
 - o **Atomicidad:** T se ejecuta completamente o no se ejecuta por completo (todo o nada)
 - o **Consistencia:** T transforma un estado consistente de la BD en otro estado consistente (los programas deben ser correctos)
 - o **Aislamiento:** Las T_i se ejecutan sin interferencias
 - o **Durabilidad:** Las actualizaciones a la BD serán durables y públicas
- Historias (Schedules)
 - o Si $T = \{T_1, T_2, \dots, T_n\}$ es un conjunto de transacciones, entonces una historia (o schedule) H sobre T es: **$H = U_{i=1, n} T_i$** , donde H respeta el orden de las acciones de cada T_i .
 - o Equivalencia: H_1 y H_2 son equivalentes si:
 - Están definidas sobre el mismo conjunto de transacciones
 - Las operaciones conflictivas tienen el mismo orden
 - Dos operaciones de T_i y T_j ($i < j$) son conflictivas si operan sobre el mismo ítem y al menos alguna de las dos es un write
 - o Historias Seriales (HS): H es serial (Hs) si para todo par de transacciones T_i y T_j en H, todas las operaciones de T_i preceden a las de T_j o viceversa.
 - Las historias seriales (y las equivalentes a estas) son las que consideraremos como correctas
 - o Historias Serializables (SR): H es serializable si es equivalente a una Historia Serial
 - o Grafo de Precedencia (SG): Dado H sobre $T = \{T_1, T_2, \dots, T_n\}$, un SG para H, $SG(H)$, es un grafo dirigido cuyos nodos son los T_i y cuyos arcos $T_i \rightarrow T_j$ ($i < j$), tal que alguna operación de T_i precede y conflictúa con alguna operación de T_j en H
 - Construcción:
 - 1) Hacer un nodo por cada T_i
 - 2) Si alguna operaciones de T_i precede y conflictua con alguna operación de T_j ($i < j$) en H, hacer un arco desde T_i a T_j
 - o **Teorema de la Seriabilidad (1):** H es serializable si y solo si su grafo de precedencia es acíclico
H es equivalente a cualquier Hs serial que sea un ordenamiento topológico de $SG(H)$
 - Algoritmo para obtener historias equivalentes:
 - 1) Buscamos el nodo al cual no llegan arcos, lo listamos y quitamos del grafo
 - 2) Luego hacemos lo mismo con todas las posibilidades que tengamos (es decir, si podemos sacar dos, hacemos 2 ramas posibles)

Locking

- Definición: El lock es un privilegio de acceso a ítem de la BD
 - o Decimos que una T_i setea u obtiene un lock sobre el ítem X
 - o Al usar locking aparecen dos problemas: Livelocks y Deadlocks
- Locking Binario (Exclusivos): Poseen 2 valores, o bloqueados o desbloqueados
 - o Construcción del Grafo de Precedencia
 - Hacer un nodo por cada T_i
 - Si T_i hace Lock de X y luego T_j ($i < j$) hace Lock de X en H, hacer un arco de T_i a T_j
- Locking Ternario (Compartidos/Exclusivos): Poseen 3 valores, Read Locked, Write Locked, y Unlocked
 - o Permiten mayor concurrencia ya que en el caso de Read Lock se permite que otros también puedan solicitar Read Locks y leer al mismo tiempo
 - o Si un Read Lock pasara a ser Write Lock lo llamaremos *lock conversion*
 - o Construcción del Grafo de Precedencia
 - Hacer un nodo por cada T_i
 - Si T_i hace RLock o WLock de X, y luego T_j ($i < j$) hace WLock de X en H, hacer un arco $T_i \rightarrow T_j$
 - Si T_i hace WLock de X y T_j ($i < j$) hace RLock de X en H, hacer un arco $T_i \rightarrow T_j$
- Modelos de Transacción basados en Locking:
 - o En este modelo una transacción T es vista como una secuencia de Locks y Unlocks
 - o Abstraeremos las operaciones del modelo anterior sin locking
- Reglas de legalidad: H es Legal si:
 - o Una T_i no puede leer ni escribir un ítem X hasta tanto no haya hecho un lock de X
 - o Una T_i que desea obtener un lock sobre X que ha sido lockeado por T_j en un modo que conflictua, debe esperar hasta que T_j haga unlock de X
- **Protocolo 2PL (Two Phase Locking):** T es 2PL si todos los locks preceden al primer unlock
- **Teorema de la Seriabilidad (2):** Dado $T = \{T_1, T_2, \dots, T_n\}$, si toda T_i en T es 2PL, entonces todo H legal sobre T es SR

Recuperabilidad

- Decimos que T_i lee de T_j en H si T_j es la T que escribió última sobre X, pero no abortó, al tiempo que T_i lee X
- Before Image: La imagen anterior de una operación Write(X, val) es el valor que tenía X justo antes de esta operación
- Clasificación de Historias según Recuperabilidad
 - o Historias Recuperables: Decimos que H es recuperable (RC), si cada transacción hace su commit después de que hayan hecho commit todas las transacciones (otras que si misma) de las cuales lee
 - o Historias que evitan aborts en cascada: Decimos que H evita aborts en cascada (avoids cascading aborts) (ACA), si cada transacción puede leer solamente aquellos valores que fueron escritos por transacciones que ya hicieron commit (o por si misma)
 - o Historias Estrictas: Decimos que H es estricta (ST), si ningún ítem X puede ser leído o sobrescrito hasta que la transacción que previamente escribió X haya finalizado haciendo o commit o abort
- **Teorema de la Recuperabilidad:** ST está contenido en ACA que a su vez está contenido en RC
 - o Nos dice que ST es más restrictivo que ACA y que RC
 - o Este concepto es ortogonal con respecto a la seriabilidad (podría suceder cualquier combinación)
 - o Hay una variante del 2PL que además de garantizar seriabilidad, garantiza recuperabilidad
- Protocolo 2PL Estricto: T cumple con 2PL estricto (2PLE) si cumple con 2PL y además no libera ninguno de sus locks exclusivos (WriteLocks) hasta después de haber hecho commit o abort
 - o Este protocolo garantiza historias estrictas (ST) con respecto a recuperabilidad, y serializables (SR) con respecto a serializabilidad, o sea que todo H que cumpla con 2PL estricto (todas sus T_i son 2PL estrictas) es **ST y SR**
 - o Sin embargo, 2PL estricto no garantiza que estemos libres de deadlocks

Logging

- Introducción
 - o ¿Qué hacen los logs?
 - Registran cambios a la BD como resultado de transacciones o acciones internas del servidor
 - o ¿Para qué sirven los logs?
 - Protegen la BD de la pérdida de integridad en casos de fallos causados por suministro eléctrico, errores en discos duros, y otros
 - o ¿Cómo funcionan los logs?
 - Trabajan de manera cíclica. Si un archivo online se llena LGWR pasará al siguiente grupo de log en el cual se produce una operación de punto de control (check point), la información es almacenada en el archivo de control (control file)

Undo

- Registros: Poseen el formato <T, x, v> donde T es la transacción, X el dato y V el valor **anterior**
- Reglas
 - o Los registros <T, x, v> se escriben a disco **antes** que el nuevo valor de x se escriba a disco en la base de datos
 - o Los registros <Commit T> se escriben a disco **después** que todos los elementos modificados por T se hayan escrito en disco
- Variantes en la recuperación
 - o Sin checkpoints
 - Dividir las transacciones en comiteadas y no comiteadas
 - A las comiteadas las ignoramos porque sabemos que ya se habían bajado a disco
 - Luego recorremos el log desde el registro más reciente hasta el último grabando el valor anterior
 - Finalmente agregamos el <Abort T> en los casos que ya terminamos de deshacer
 - o con Checkpoint Quiescente
 - La técnica anterior tiene el problema de que si pasa algo hay que volver hasta el principio del log
 - Entonces, primero dejamos de aceptar transacciones nuevas
 - Luego dejamos que terminen las transacciones activas
 - Escribimos el registro <CKPT> y efectuamos un flush, para luego aceptar nuevamente transacciones
 - Con esto arreglamos el problema de tener que recorrer todo el log, sin embargo hay que parar la BD
 - o con Checkpoint no-Quiescente
 - La política para no tener que parar la BD, es con esta variante
 - 1ero escribir un registro <Start CKPT(T1,T2,...,Tk)> en el log, y efectuar un flush. T1,T2,...,Tk son las T activas (aquellas con <START T> y sin <Commit T>) al momento de introducir el checkpoint
 - 2do esperar a que todas las transacciones T1,T2,...,Tk terminen (ya sea abortando o comiteando)
 - Por último escribir el registro <End CKPT> en el log y efectuar un flush
 - Luego para buscar hasta donde leer el log, vamos hasta el último End CKPT

Redo

- Registros: Poseen el formato <T, x, v> donde T es la transacción, X el dato y V el valor **posterior**
- Reglas
 - o Los registros <T,x,v> se escriben a disco **antes** que el nuevo valor de x se escriba a disco en la base de datos
 - o Los registros <Commit T> se escriben a disco **antes** que todos los elementos modificados por T se hayan escrito en disco.

- Variantes en la recuperación
 - o Sin checkpoints
 - Dividimos las transacciones en comiteadas y no comiteadas, igual que en **Undo**
 - Las no comiteadas las podemos ignorar, ya que estamos seguros por la regla dos que sus datos nunca llegaron a disco. En cambio, para las comiteadas, no estamos seguros que sus cambios se hayan registrado en la BD, entonces hay que rehacer las operaciones
 - Nos ubicamos al comienzo del archivo y vamos rehaciendo las que vimos comiteadas
 - o con Checkpoint Quiescente
 - Si bien podemos esperar a que comiteen las transacciones para poner un CKPT, no sabemos si bajan a disco. Modificamos la variante del Undo con lo siguiente:
 - Dejar de aceptar nuevas transacciones
 - Esperar a que las modificaciones realizadas por las transacciones ya comiteadas sean escritas a disco
 - Escribir un registro <CKPT> en el log y luego efectuar un flush, para luego aceptar nuevas trans.
 - Ahora no importa que las transacciones activas terminen. Se espera a que las comiteadas bajen!
 - o con Checkpoint no-Quiescente
 - Posee el mismo problema de parar la base, entonces:
 - 1ero escribir un registro <Start CKPT(T1,T2,...,Tk)> en el log, y efectuar un flush. T1,T2,...,Tk son las T activas (aquellas con <START T> y sin <Commit T>) al momento de introducir el checkpoint
 - Esperar a que todas las modificaciones realizadas por T ya comiteadas al momento de introducir el Start CKPT sean escritas a disco. Escribir el registro <End CKPT> en el log y efectuar un flush
 - Ahora si encontramos un <End CKPT>, podemos ignorar todas aquellas transacciones que comitearon previamente al último registro Start CKPT

Undo/Redo

- Registros: Poseen el formato <T, X, v, w> donde T y X son iguales al caso ant. y V es valor **anterior**, W el **posterior**
- Regla: Los registros <T,x,v,w> se escriben a disco antes que el nuevo valor de x se escriba a disco en la BD
- Es más flexible de los tres, el objetivo es deshacer las transacciones no comiteadas y rehacer las comiteadas, esto es, combinar los mecanismos de **Undo y Redo**
- Variantes
 - o Sin checkpoints
 - 1ero deshacer las transacciones no comiteadas desde el último registro hasta el principio
 - 2do rehacer las transacciones comiteadas desde el principio hasta el final
 - 3ero se agrega un registro abort para cada transacción no comiteada, y se hace un flush del log
 - o con checkpoints quiescente
 - Dejar de aceptar nuevas transacciones
 - Esperar a que todas las modificaciones realizadas por alguna T (comiteada o no) sea escrita a disco
 - Escribir un registro <CKPT> en el log y luego efectuar un flush, para aceptar nuevas transacciones
 - No ganamos nada con las transacciones a deshacer, si alguna no comiteo tenemos que retroceder todo. Sin embargo ganamos en las rehacer, porque vamos desde el último CKPT
 - o con checkpoints no quiescente
 - Escribir un registro <Start CKPT(T1,T2,...,Tk)> en el log, y efectuar un flush. T1,T2,...,Tk son las T activas (aquellas con <START T> y sin <Commit T>) al momento de introducir el checkpoint
 - Esperar a que todas las modificaciones realizadas por transacciones (comiteadas o no) al momento de introducir el Start CKPT sean escritas a disco
 - Escribir el registro <End CKPT> en el log y efectuar un flush
 - En el deshacer de las transacciones no comiteadas debemos ir hasta su Start
 - Sin embargo en el rehacer vamos desde el start del último CKPT que encontremos, como siempre debemos agregar los aborts en los casos correspondientes

Mapeo Objeto Relacional

Motivación: Un modelo orientado a objetos posee gran versatilidad en cuanto a los tipos de datos que puede contener, mientras que un modelo relacional soporta queries de gran nivel. Las bases de datos relacionales-orientadas a objetos poseen lo mejor de cada mundo.

- Tipos de Datos Complejos: la ventaja es que permite dominios no atómicos. Es más intuitivo para modelado con datos complejos.
 - o Retiene el fundamento matemático del modelo relacional.
 - o Viola la 1er forma normal.
 - o En 1999 fueron incluidas muchas extensiones para el SQL que la mayoría de los motores comerciales tienen incluidas, estas permiten collections, structs, herencia, referencia entre objetos (llamadas con “->”) y constructores.
 - Las herencias múltiples no están estandarizadas en los SQL de 1999 y 2003.
 - o Se pueden utilizar tanto para definir una tabla como un atributo de un campo.
 - o Si creamos tipos de conjuntos, se pueden desdoblar. La transformación de una relación anidada en una forma con menos (o ninguna) relación con valores de atributos se denomina unnesting (desanidamiento).
 - Desde ya, también se puede hacer lo contrario: nesting (anidamiento).
 - o Ejemplo 1: Si cada libro tiene un título, un array de autores, un array de quienes lo publican y un set de palabras clave, llenar esta estructura equivale a un registro.
 - o Ejemplo 2:
 - create table person (name Name, address Address, dateOfBirth date), donde:
 - create type Name as (firstname varchar(20), lastname varchar(20)) final
 - create type Address as (street varchar(20), city varchar(20), zipcode varchar(20)) not final
 - o Ejemplos 3: Para crear tablas o atributos basados en estos tipos:
 - create table customer of CustomerType
 - create table person_r(name **row**(firstname varchar(20),lastname varchar(20)),address **row**(street varchar(20),city varchar(20),zipcode varchar(20)),dateOfBirth date)
 - o Ejemplo 4: Creación y utilización de un array:
 - Creación:
 - create type Publisher as (name varchar(20), branch varchar(20));
 - create type Book as (title varchar(20), author_array varchar(20) array [10], pub_date date, publisher Publisher, keyword-set varchar(20) multiset);
 - create table books of Book;
 - Utilización:
 - insert into books values ('Compilers', array['Smith','Jones'], new Publisher ('McGraw-Hill','New York'), multiset ['parsing','analysis']);
 - o Ejemplos 5: Desdoblamiento y anidamiento de listas
 - select author_array[1], author_array[2], author_array[3] from books where title = 'Database System Concepts'
 - select B.title, A.author, A.position from books as B, unnest (B.author_array) with ordinality as A (author, position)
 - select title, author, Publisher (pub_name, pub_branch) as publisher, collect (keyword) as keyword_set from flat_books group by title, author, publisher
 - select title, collect (author) as author_set, Publisher (pub_name, pub_branch) as publisher, collect (keyword) as keyword_set from flat_books group by title, publisher
 - select title, array (select author from authors as A where A.title = B.title order by A.position) as author_array, Publisher (pub-name, pub-branch) as publisher, multiset (select keyword from keywords as K where K.title = B.title) as keyword_set from books4 as B

XML

Definiciones iniciales:

- Datos estructurados: poseen una representación estricta.
- Datos semiestructurados:
 - o Poseen una “cierta” estructura, no toda la información recolectada tendrá la misma.
 - o Es esquema está mezclado con el dato

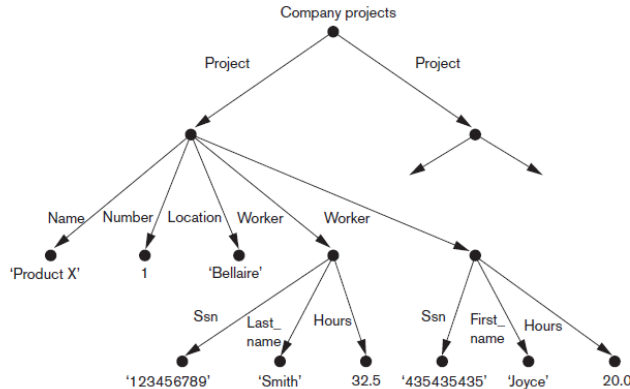


Figure 12.1
Representing
semistructured data
as a graph.

- o
- Datos autodescriptivos: pueden ser mostrados directamente. Tienen labels o tags que representan los nombres y tipos de los atributos.
- Datos no-estructurados: Indicación delimitada desde el documento de los datos que contienen información incrustada dentro de ella.
- XML: Extensible Markup Language, tags html con información, como dice su nombre es “extensible”, es decir que se pueden crear cualquier tipo de tags.
- DOM: Document Object Model, manipula el XML bien formado como un árbol.
- SAX: Simple API for XML, procesa XML en el aire, ideal para streaming.

Motivación:

- Uno de los usos principales del XML es el **intercambio de datos** cada vez más crítico en el mundo.
- Cada área podría definir sus propios estándar para dicho intercambio.
- La definición de la estructura podría estar en un DTD (Document Type Descriptor) o en un Esquema XML (nuevo).
 - o Las novedades con el Esquema XML son entre otras: los tipos de datos, constraints, datos complejos, FK, que está especificado en un XML (se especifica a si mismo), posee un namespace (identificador), con la contra que todas estas ventajas poseen un mayor costo de mantenimiento.
- Este modelo es fácil de generar y leer.
- Cuando uno exporta un XML no muestra su modelo de datos, y entre ellos sus dependencias o restricciones.

Algunas características extra:

- Restricciones:
 - o Name es un atributo
 - o Base es el dominio general de restricción (ej xs:Integer)
 - o xs:restriction es un subelemento
 - o xs:{min, max}{Inclusive, Exclusive} son atributos de filtro
 - o xs:enumeration es un valor para enumeración de valores

```
<xs:simpleType name = "license">
  <xs:restriction base = "xs:string">
    <xs:enumeration value = "Full" />
    <xs:enumeration value = "Beer only" />
    <xs:enumeration value = "Sushi" />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name = "price">
  <xs:restriction
    base = "xs:float"
    minInclusive = "1.00"
    maxExclusive = "5.00" />
</xs:simpleType>
```

Realización de consultas y transformación de los XML:

- Los lenguajes estándares son:

- XQuery (para consultas ricas y completas). Estos lenguajes están basados en la búsqueda arbórea.
 - Equivalencias con el SQL:
 - for = SQL from; where = SQL where; order by = SQL order by; result = SQL select
 - let permite variables temporales, no tiene equivalencia en SQL
 - Tiene foreach, while y otras operaciones.

• Ejemplo de foreach:

- for \$beer in document("bars.xml")/BARS/BEER/@name
 - return <BEERNAME> {\$beer} </BEERNAME>
- Devuelve los atributos "name":
 - <BEERNAME>Bud</BEERNAME>
 - <BEERNAME>Miller</BEERNAME>

• Ejemplo de let:

- let \$d := document("bars.xml"), let \$beers := \$d/BARS/BEER/@name
- return <BEERNAMES> {\$beers} </BEERNAMES>
- Devuelve todos los atributos nombres consecutivos:
 - <BEERNAMES>Bud Miller ...</BEERNAMES>

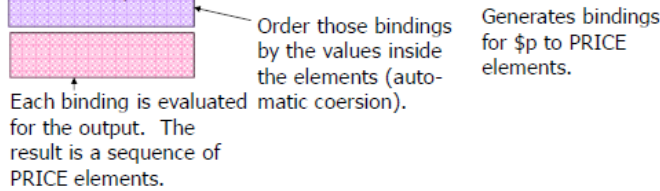
• Ejemplo de consulta con order by:

- List all prices for Bud, lowest first.

let \$d := document("bars.xml")

```
for $p in $d/BARS/BAR/PRICE[@theBeer="Bud"]  
order by $p
```

return \$p



• Ejemplo de distinct:

- return distinct-values(let \$bars = doc("bars.xml") return \$bars/BARS/BAR/PRICE)

- XPath (simple para expresiones de búsqueda) (con muchos "shorcuts" de expresiones más complejas, ej: en lugar de PADRE/child::HIJO iría PADRE/HIJO)

- Si el árbol sigue pero la consulta termina se devuelve un gran elemento con todos sus hijos

```
<BARS>  
  <BAR name = "JoesBar">  
    <PRICE theBeer = "Bud">2.50</PRICE>  
    <PRICE theBeer = "Miller">3.00</PRICE>  
  </BAR> ...  
  <BEER name = "Bud" soldBy = "JoesBar"  
    SuesBar ... "> ...  
</BARS>
```

One item, the BARS element

- Ej: "/university/instructor/name" devuelve los nombres
- Ej: "/university/instructor/name/@NombreAtributo" devuelve los valores de dicho atributo
- Ej: "//..." comienza desde cualquier path, también se utilizan "*/*..." y "ALGO/*".
- Condicionales: "/BARS/BAR/PRICE[. < 2.75]" se fija el valor dentro de PRICE
- XSLT (diseñado para la transformación)
 - Son expresados con reglas llamadas "templates", que combinan la selección utilizando XPath

Data Warehousing

Definición: En el contexto de la informática, un almacén de datos (del inglés data warehouse) es una colección de datos orientada a un determinado ámbito (empresa, organización, etc.), integrado, no volátil y variable en el tiempo, que ayuda a la toma de decisiones en la entidad en la que se utiliza. Se trata, sobre todo, de un expediente completo de una organización, más allá de la información transaccional y operacional, almacenado en una base de datos diseñada para favorecer el análisis y la divulgación eficiente de datos. El almacenamiento de los datos no debe usarse con datos de uso actual. Los almacenes de datos contienen a menudo grandes cantidades de información que se subdividen a veces en unidades lógicas más pequeñas dependiendo del subsistema de la entidad del que procedan o para el que sean necesario.

Bill Inmon fue uno de los primeros autores en escribir sobre el tema de los almacenes de datos, define un data warehouse (almacén de datos) en términos de las características del repositorio de datos:

- **Orientado a temas:** Los datos en la base de datos están organizados de manera que todos los elementos de datos relativos al mismo evento u objeto del mundo real queden unidos entre sí.
- **Variante en el tiempo:** Los cambios producidos en los datos a lo largo del tiempo quedan registrados para que los informes que se puedan generar reflejen esas variaciones. Todos los datos del datawarehouse refieren a un particular momento en el tiempo (como una “foto” o “snapshot”).
- **No volátil:** La información no se modifica ni se elimina, una vez almacenado un dato, éste se convierte en información de sólo lectura, y se mantiene para futuras consultas. Esto permite el análisis retrospectivos sobre la marcha del negocio.
- **Integrado:** La base de datos contiene los datos de todos los sistemas operacionales de la organización, y dichos datos deben ser consistentes.

Comparación: Datos Operacionales Vs Data Warehouse

	Datos operacionales	Data Warehouse
Contenido	Valores elementales	Datos sumariados, derivados
Organización	Por aplicación	Por tema
Estabilidad	Dinámicos	Estáticos hasta su actualización

	Datos operacionales	Data Warehouse
Uso	Predecible Repetitivo	Ad hoc Heurístico
Tiempo de respuesta	Segundos	Segundos a minutos
Cantidad de registros involucrados	A lo sumo decenas	Cientos - miles

	Datos operacionales	Data Warehouse
Estructura	Optimizada para uso transaccional (NORMALIZADA)	Optimizada para queries complejos (DESNORMALIZADA)
Frecuencia de acceso	Alta	Media y baja
Tipo de acceso	Lectura / escritura Actualización campo por campo	Lectura Sumarización

- Arquitectura de los datos



- Problemáticas de los datos:

○ Cuando son demasiados:

- datos corruptos o con ruido
- datos redundantes (requieren factorización)
- datos irrelevantes
- excesiva cantidad de datos

○ Pocos datos

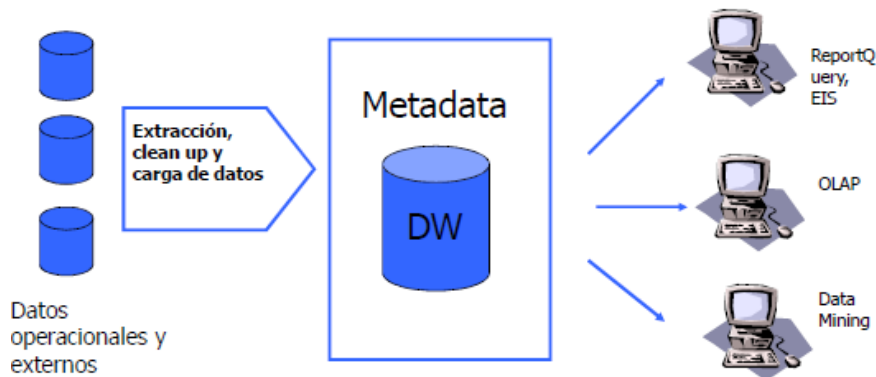
- atributos perdidos (missings)
- valores perdidos
- poca cantidad de datos

○ Datos fracturados

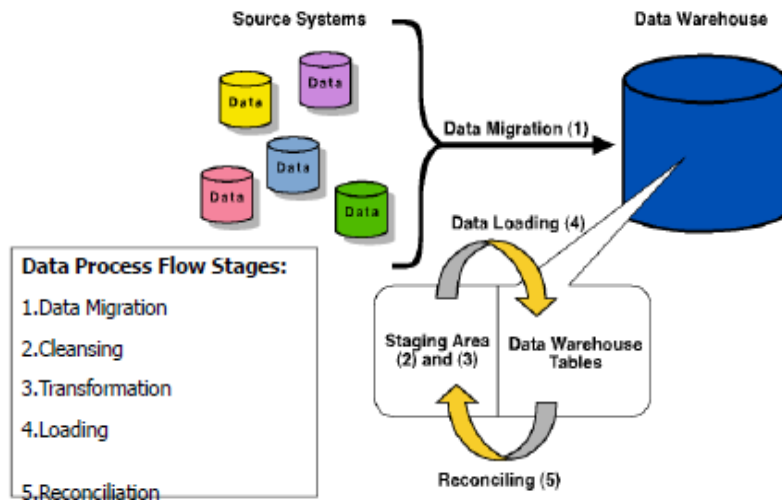
- datos incompatibles
- múltiples fuentes de datos

- Data Marts: Técnicamente es un subconjunto del DW orientado a una finalidad específica de negocio : marketing, finanzas, producción, etc. El término se utiliza también para identificar soluciones alternativas a un DW corporativo más reducidas y de menor costo y tiempo de implantación.

- Explotación del Datawarehouse



- ETL: Extract, Transform, Load



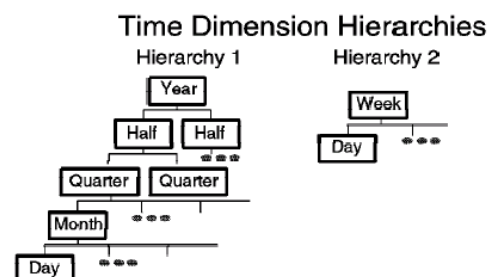
-
- Procedimientos (herramientas) destinados a obtener los datos de las fuentes operacionales, limpiarlos, convertirlos a los formatos de utilización y cargarlos en el repositorio final.
- Etapas:
 - Migración
 - Staging area : área de trabajo fuera del DW
 - El propósito de la migración es mover los datos de los sistemas operacionales a las áreas de trabajo (staging areas).
 - NO se debe mover datos innecesarios (control preventivo).
 - Limpieza
 - Corregir, estandarizar y completar los datos
 - Identificar datos redundantes
 - Identificar valores atípicos (outliers)
 - Identificar valores perdidos (missings)
 - Las denominaciones de los sistemas operacionales deben uniformarse y referenciarse con nombres propios de los sistemas de negocios (autodocumentados)
 - Los tipos de dato asociados a cada atributo deben estandarizarse y consolidarse para las diferentes fuentes
 - Se debe uniformar las tablas de códigos de los sistemas operacionales y simplificar esquemas de codificación
 - Datos complejos, que representan varios atributos a la vez, deben ser particionados.
 - Ejemplos:
 - Eliminar transacciones con monto = 0 (promociones, regalos)
 - Eliminar transacciones anuladas (balance = 0)
 - Normalizar nombres de marcas de auto, de direcciones, etc.
 - Eliminar fechas de nacimiento inválidas (edad > 100 años o negativa)

- Transformación (cálculos, agregados, normalización, etc.)
 - Metadata: Provee a los usuarios de información para facilitarles el acceso e interpretación del contenido del DW
 - Información
 - Fuentes de datos
 - Descripción de operaciones de transformación
 - Estructura de datos del DW
 - Reglas de clean up
 - Referencias históricas y temporales, etc.
 - Importancia:
 - Los metadatos proveen la vinculación entre los datos y los usuarios de negocio. Describen los datos
 - Incluyen los modelos lógicos de datos, el mapeo de los datos a los sistemas transaccionales, el esquema físico de los datos, información de carga, actualización y seguridad, etc.
 - Procesos destinados a adaptar los datos al modelo lógico del DW
 - Se generan “reglas de transformación”
 - Las reglas deben validarse con los usuarios del DW
 - Generalmente el DW no contiene información de las entidades que - en los sistemas operacionales - son muy dinámicas y sufren frecuentes cambios.
 - Si es necesario se utilizan Snapshots (fotos instantáneas)
 - La des-normalización de los datos tiene como propósito mejorar la performance.
 - Otro propósito es el de reflejar relaciones estáticas, es decir, que no cambian en una perspectiva histórica. Por ejemplo: producto - precio vigente al momento de facturación.
 - Sumarizaciones: Muestran los datos de una manera más resumida, permitiendo, precisamente, calcular valores agregados, que no son los datos directos registrados, sino datos derivados de ellos. Se puede considerar, en cierto modo, una generalización de los datos y, por tanto, suele facilitar el aprendizaje. Pero no es sólo una cuestión de eficiencia, sino, muchas veces, una necesidad. En la mayoría de eventos físicos, cuando más se detallan los datos menos patrones suelen encontrarse
 - Los datos sumarizados aceleran los tiempos de análisis
 - Las sumarizaciones también ocultan complejidad de los datos
 - Las sumarizaciones pueden incluir joins de múltiples tablas
 - Las sumarizaciones proveen múltiples vistas del mismo conjunto de datos detallados (dimensiones)
 - Mantenimiento:
 - El mantenimiento de las sumarizaciones es una tarea crítica
 - El DW debe actualizarlas a medida que se cargan nuevos datos
 - Debe existir alguna forma de navegar los datos hasta el nivel de detalle (drill down)
 - La definición de la granularidad es un problema serio de diseño
 - Como definir tiempo, entidades, etc.
- Carga
 - Soporte físico de los datos (DBMS)
 - Aproximaciones:
 - Full Refresh (Carga completa) – Difícil de realizar
 - Incremental (Carga incremental)

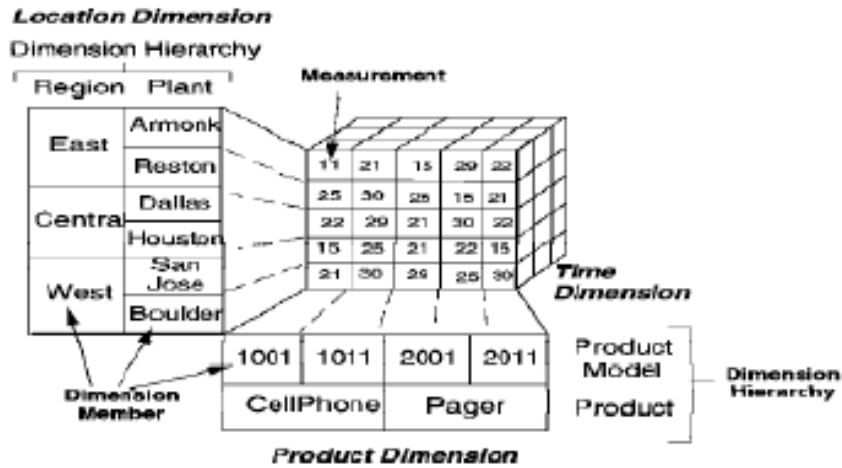
- Conciliación – Validación
 - Integridad de datos: es cuando se ajustan a todos los estándares de completitud.
 - La idea es que: Todos los datos del DW sean **correctos**, y **completos**
 - Si esto no pasa, los resultados no son creíbles y dejarán de usar el sistema
 - Se utilizan controles de prevención antes de la carga y de detección una vez ya cargados
 - Maneras:
 - Completa (al final del proceso)
 - Por etapas a medida que se cargan los datos
 - Los controles incluyen reportes que comparan los datos del DW con las fuentes operacionales a través de totales de control, número de registros cargados, valores originales vs valores limpios (transformados), etc.
- Herramientas
 - OLAP, reporting, Data Mining, etc.
 - OLAP (On-Line Analytical Processing): Es una solución utilizada en el campo de la llamada Inteligencia empresarial (o Business Intelligence) cuyo objetivo es agilizar la consulta de grandes cantidades de datos. Para ello utiliza estructuras multidimensionales (o Cubos OLAP) que contienen datos resumidos de grandes Bases de datos o Sistemas Transaccionales (OLTP)
 - Pueden ser procesos manuales diseñados a medida (queries SQL, programas en Visual Basic, etc.).
 - Existen herramientas que proporcionan interfaces visuales para definir joins, transformaciones, agregados, etc. sobre las plataformas más comunes.

Modelado de Datos

- Claves:
 - Visualización del universo del negocio
 - Modelo de abstracción de las “preguntas” que los usuarios necesitan responder
 - Diseño del plan de implantación del Data Warehouse
- Técnicas:
 - Modelo ER: Entidades, Relaciones, Atributos
 - Modelo Dimensional: Hechos, Dimensiones, Medidas
 - **Hechos:** Colección de ítems de datos y datos de contexto. Cada hecho representa un ítem de negocio, una transacción o un evento. Se registran en las tablas CENTRALES del DW
 - **Dimensión:** Colección de miembros o unidades o individuos del mismo tipo
 - Cada punto de entrada de la tabla de HECHOS está conectado a una DIMENSION
 - Determinan el contexto de los HECHOS
 - Se utilizan como parámetros para los análisis OLAP
 - Dimensiones habituales y sus miembros son:
 - Tiempo (Meses, Trimestres, Años)
 - Geografía (País, Región, Ciudad)
 - Cliente (Id Cliente)
 - Vendedor (id Vendedor)
 - **Medida:** Es un atributo numérico de un hecho que representa la performance o comportamiento del negocio relativo a la dimensión
 - Ejemplos: Ventas en \$\$, Cantidad de productos, Total de transacciones, etc.



- Visualización:



- Anotaciones:
 - o El modelo dimensional es ideal para soportar las 4 operaciones básicas de la tecnología OLAP:
 - Relacionadas con la granularidad: ROLL UP (Te alejás en la dimensión) - DRILL DOWN (Te enfocás en algo de la dimensión)
 - Navegación por las dimensiones : SLICE – DICE

\$ of Anheuser-Busch by drinker/bar

	Jim	Bob	Mary
Joe's Bar	45	33	30
Nut-House	50	36	42
Blue Chalk	38	31	40

Roll up
by Bar

\$ of A-B / drinker

Jim	Bob	Mary
133	100	112

Drill down
by Beer

\$ of A-B Beers / drinker

	Jim	Bob	Mary
Bud	40	29	40
M'lob	45	31	37
Bud Light	48	40	35

62

Volume of Prod (numbers in 1000)		1996			
		Qtr 1	Qtr 2	Qtr 3	Qtr 4
West	San Jose	78	45	34	56
	Boulder	90	67	87	91

Roll Up
Dimension: Time

Volume of Prod (numbers in 1000)		Quarter 1		
		Jan	Feb	Mar
West	San Jose	30	26	22
	Boulder	28	30	32

Volume of Prod (numbers in 1000)		CellPhone		Pager	
		1001	1011	2001	2011
West	San Jose	33	12	8	12
	Boulder	45	34	20	23

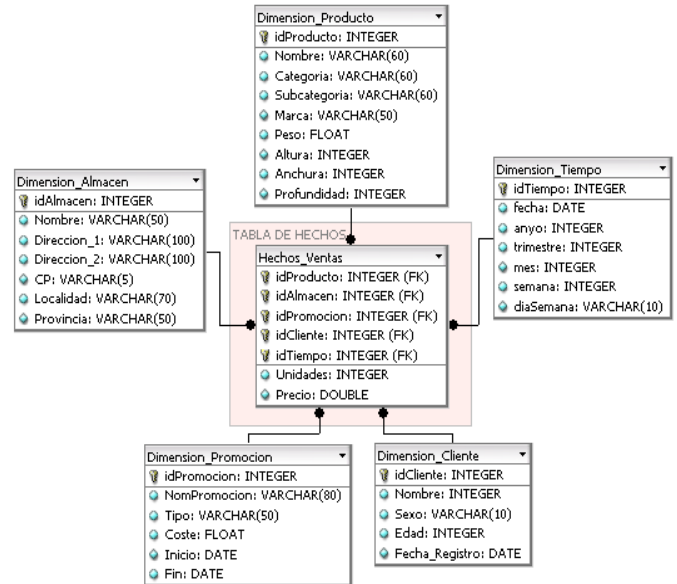
Drill-Down
Dimension: Location
Member: San Jose

Volume of Prod (numbers in 1000)		CellPhone		Pager	
		1001	1011	2001	2011
San Jose	Team1	20	8	6	7
	Team2	13	4	2	5

- Slice: Es un subconjunto del “array multidimensional” que tiene un único valor para una o más dimensiones.
Es recortar una “rebanada” del cubo
- Dice: Es como el “slice” pero para 2 o más valores de una o más dimensiones
- Modelos básicos dimensionales:

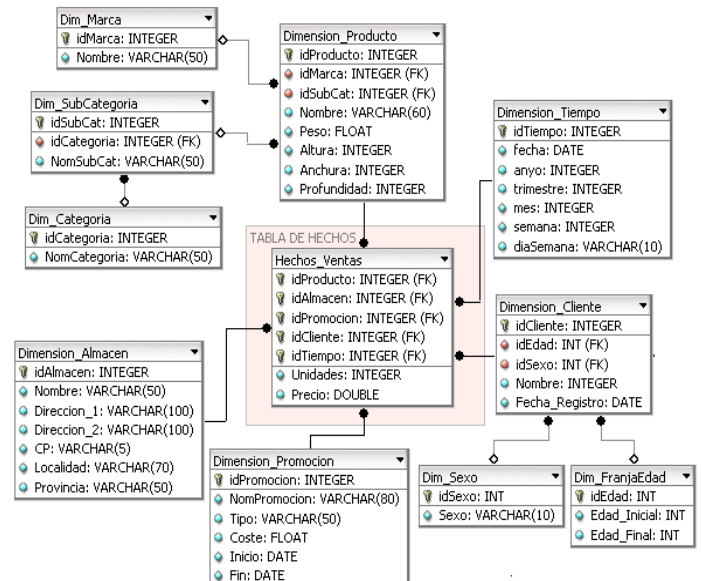
- Star:

- En las bases de datos usadas para data warehousing, un esquema en estrella es un modelo de datos que tiene una tabla de hechos (o tabla fact) que contiene los datos para el análisis, rodeada de las tablas de dimensiones. Este aspecto, de tabla de hechos (o central) más grande rodeada de radios o tablas más pequeñas es lo que asemeja a una estrella, dándole nombre a este tipo de construcciones
- Las tablas de dimensiones tendrán siempre una clave primaria simple, mientras que en la tabla de hechos, la clave principal estará compuesta por las claves principales de las tablas dimensionales



- Snowflake:

- Es una estructura algo más compleja que el esquema en estrella. Se da cuando alguna de las dimensiones se implementa con más de una tabla de datos. La finalidad es normalizar las tablas y así reducir el espacio de almacenamiento al eliminar la redundancia de datos; pero tiene la contrapartida de generar peores rendimientos al tener que crear más tablas de dimensiones y más relaciones entre las tablas (JOINS) lo que tiene un impacto directo sobre el rendimiento



- Posibles relaciones con el modelo ER:

- Foreign keys → Dimensión
- Hecho → Entidad

Data Mining

Definición: La minería de datos (es la etapa de análisis de "Knowledge Discovery in Databases" o KDD), es un campo de las ciencias de la computación referido al **proceso que intenta descubrir patrones en grandes volúmenes de conjuntos de datos**. Utiliza los métodos de la inteligencia artificial, aprendizaje automático, estadística y sistemas de bases de datos. El objetivo general del proceso de minería de datos consiste en extraer información de un conjunto de datos y transformarla en una estructura comprensible para su uso posterior. Además de la etapa de análisis en bruto, que involucra aspectos de bases de datos y gestión de datos, procesamiento de datos, el modelo y las consideraciones de inferencia, métricas de Intereses, consideraciones de la Teoría de la complejidad computacional, post-procesamiento de las estructuras descubiertas, la visualización y actualización en línea.

La tarea de minería de datos real es el análisis automático o semi-automático de grandes cantidades de datos para extraer patrones interesantes hasta ahora desconocidos, como los grupos de registros de datos (análisis cluster), registros poco usuales (la detección de anomalías) y dependencias (minería por reglas de asociación). Esto generalmente implica el uso de técnicas de bases de datos como los índices espaciales. Estos patrones pueden entonces ser visto como una especie de resumen de los datos de entrada, y puede ser utilizado en el análisis adicional o, por ejemplo, en la máquina de aprendizaje y análisis predictivo. Por ejemplo, el paso de minería de datos podrían identificar varios grupos en los datos, que luego pueden ser utilizados para obtener resultados más precisos de predicción por un sistema de soporte de decisiones. Ni la recolección de datos, preparación de datos, ni la interpretación de los resultados y la información son parte de la etapa de minería de datos, pero que pertenecen a todo el proceso KDD como pasos adicionales.

Cómo se integra en el proceso de descubrimiento del conocimiento

- Data Mining: the core of knowledge discovery process
- Data Cleaning, Data Integration (Data Warehouse), Task Relevant Data, Data Mining, Pattern Evaluation, Knowledge
- Funcionalidades:
 - Descripción de conceptos: Caracterización y discriminación
 - Generalizar, Resumir y **contrastar las características de la información** (por ejemplo las regiones secas vs. Las regiones húmedas)
 - Asociación (correlación y causalidad)
 - Multi-dimensionales vs. única dimensión
 - $\text{age}(X, "20..29") \wedge \text{income}(X, "20..29K") \Rightarrow \text{buys}(X, "PC")$ [support = 2%, confidence = 60%]
 - $\text{contains}(T, "computer") \Rightarrow \text{contains}(x, "software")$ [1%,75%]
 - Clasificación y Predicción
 - Encontrar modelos o funciones que describan y distingan clases para futuras predicciones
 - Ej: Clasificar países de acuerdo a su clima, clientes de acuerdo a su comportamiento
 - Presentación: Árboles de decisión, reglas de clasificación, redes neuronales
 - Predicción: Predecir valores numéricos desconocidos o faltantes
 - Cluster Analysis
 - No se sabe a que clase pertenecen los datos: se agrupan datos para formar clases
 - El Clustering se basa en el principio de maximizar la similitud dentro de la clase y minimizar la misma entre clases
 - Análisis de Outliers
 - Outlier: Un dato (o un objeto) que no respeta el comportamiento general
 - Puede ser ruido o excepciones, pero son muy útiles en la detección de fraudos o eventos raros
 - Análisis de tendencias y evolución
 - Tendencia y Desvíos: Análisis de regresión
 - Análisis de patrones secuenciales
 - Análisis de similitudes
 - Otros análisis estadísticos o de patrones

- Técnicas

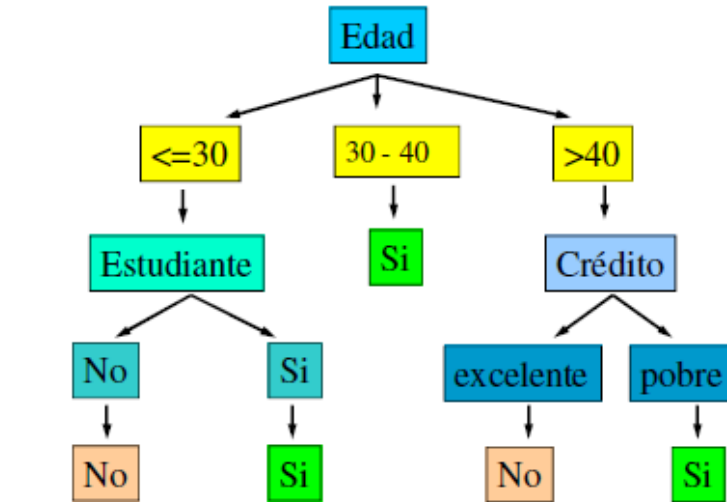
- o Supervisadas

- **Redes Neuronales**

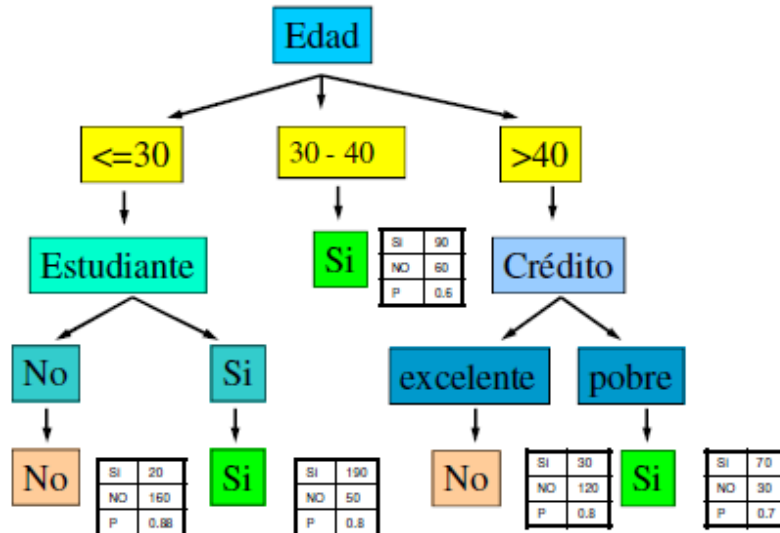
- Son sistemas
 - o Capaces de aprender, adaptarse a condiciones variantes, adaptarse al ruido, predecir el estado futuro, enfrentar problemas que eran resueltos sólo por el cerebro humano
 - No son algorítmicas
 - o No se programan haciéndoles seguir una secuencia predefinida de instrucciones
 - o Las RNA generan ellas mismas sus propias "reglas", para asociar la respuesta a su entrada
 - o Aprenden por ejemplos y de sus propios errores
 - o Utilizan un procesamiento paralelo mediante un gran número de elementos altamente interconectados
 - Para mejorar su performance las RNA pueden ser combinadas con otras herramientas
 - o Lógica Difusa (Fuzzy Logic), Algoritmos Genéticos, Sistemas Expertos, Estadísticas, Transformadas de Fourier, Wavelets
 - Biología
 - o Las Redes Neuronales Artificiales (RNA) se basan en modelos simplificados de las neuronas las reales. Las partes mas comúnmente modelados son:
 - Axón, Dendrita, Sinapsis, Cuerpo de Célula
 - Aprendizaje
 - o Para que una RNA aprenda o se entrene se deben hacer pasar a todos los valores de entrenamiento por el siguiente proceso, según la topología de la red este ciclo puede repetirse varias veces y con los datos en diferente orden
 - Calcular la diferencia de la salida con la esperada
 - Corregir los valores de los W que intervienen en esa salida de modo que se achique esa diferencia
 - Se utiliza una constante muy pequeña (Delta)
 - No se busca que la diferencia tienda a cero sino que se minimice de a poco
 - Si la constante es muy grande o se minimiza la diferencia muy de golpe se corre el riesgo de que cada vez que se aprende algo nuevo se modifique demasiado lo que aprendió anteriormente
 - Tipos:
 - o Perceptrón Multicapa: red formada por varias capas, resuelve problemas no lineales
 - o Red de Hopfield (Mapas Asociativos)
 - o Red de Kohonen (Mapas Autoorganizativos)
 - Aplicaciones
 - o La clase de problemas que mejor se resuelven con las redes neuronales son los mismos que el ser humano resuelve mejor pero a gran escala
 - Asociación, evaluación, reconocer patrones
 - o Las redes neuronales son ideales para problemas que son muy difíciles de calcular
 - No requieren de respuestas perfectas, sólo respuestas rápidas y buenas
 - o Ejemplo: Escenario Bursatil: comprar, vender, mantener
 - Fallas
 - o Cálculos precisos
 - o Procesamiento serie
 - o Reconocer nada que no tenga inherentemente algún tipo de patrón

■ Árboles de Decisión

- Los nodos internos son preguntas sobre los atributos
- Las hojas representan las etiquetas o clases resultantes
- La generación del árbol tiene fundamentalmente dos pasos
 - Construcción
 - Al comienzo todos los ejemplos están en la raíz del árbol
 - Se dividen los ejemplos en forma recursiva basado en atributos elegidos
 - Pruning
 - Identificar y remover ramas que representan outliers o ruido
- Ejemplo: Árbol de decisión para ver quien compra una computadora



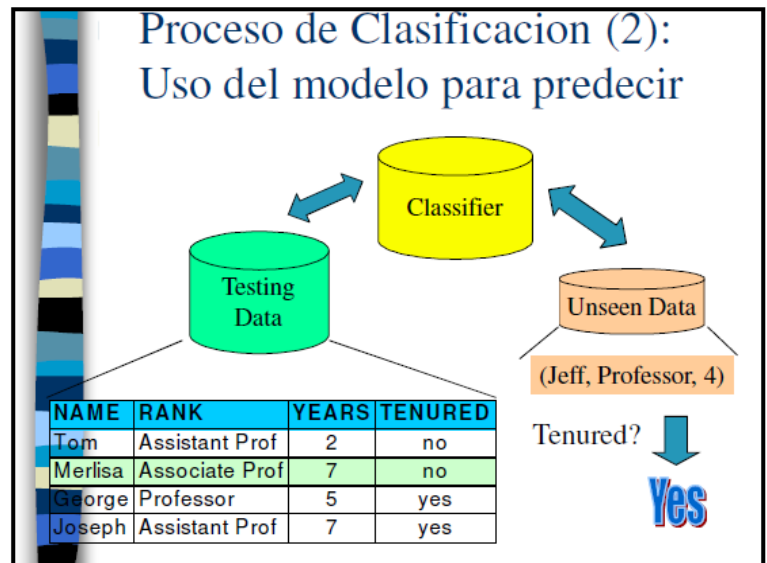
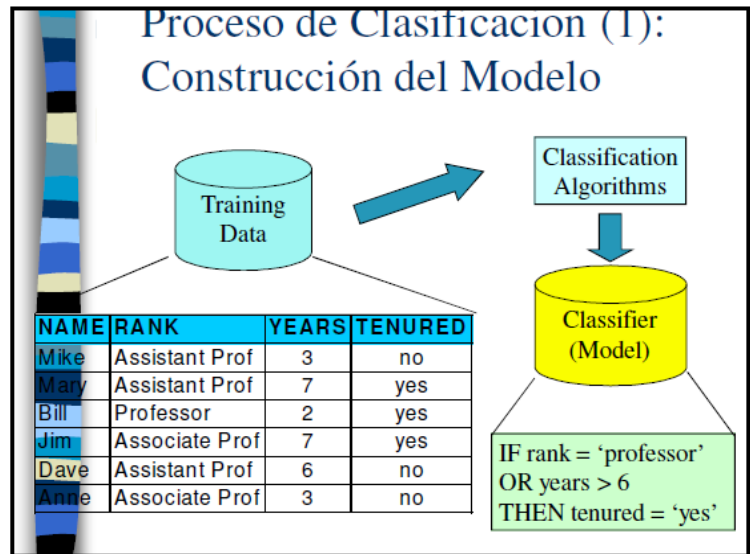
-
- Idem con probabilidad:



-
- Extracción de reglas de clasificación a partir de los árboles
 - Representa el conocimiento en la forma de reglas de IF-THEN
 - Se genera una regla para cada camino desde la raíz hasta las hojas
 - Cada par atributo – valor forma una conjunción
 - La hoja tiene la clase a predecir
 - Las reglas son fácilmente entendibles por los seres humanos
 - Ejemplos
 - IF edad = “≤30” AND estudiante = “no” THEN compra_PC = “no”
 - IF edad = “≤30” AND estudiante = “yes” THEN compra_PC = “si”, etc.

- Evitar el Overfitting en la clasificación
 - Definir Overfitting (sobreentrenamiento):
 - En aprendizaje automático, el sobreajuste (también es frecuente emplear el término en inglés overfitting) es el efecto de sobreentrenar un algoritmo de aprendizaje con unos ciertos datos para los que se conoce el resultado deseado. El algoritmo de aprendizaje debe alcanzar un estado en el que será capaz de predecir el resultado en otros casos a partir de lo aprendido con los datos de entrenamiento, generalizando para poder resolver situaciones distintas a las acaecidas durante el entrenamiento. Sin embargo, cuando un sistema se entrena demasiado (se sobreentrena) o se entrena con datos extraños, el algoritmo de aprendizaje puede quedar ajustado a unas características muy específicas de los datos de entrenamiento que no tienen relación causal con la función objetivo. Durante la fase de sobreajuste el éxito al responder las muestras de entrenamiento sigue incrementándose mientras que su actuación con muestras nuevas va empeorando
 - El árbol obtenido puede hacer overfitting sobre el conjunto de entrenamiento
 - Si hay demasiadas ramas algunas pueden reflejar anomalías
 - Como consecuencia de esto se tiene una performance muy mala sobre ejemplos nuevos
 - Dos aproximaciones para evitar el overfitting
 - Prepruning: Interrumpir la construcción del árbol en forma anticipada. No partir un nodo si la mejora que esto produce está por debajo de un cierto umbral (Es difícil encontrar el umbral adecuado)
 - Postpruning: quitar ramas de un árbol ya construido (Se puede usar un conjunto diferente del de entrenamiento para hacer esto)
- Detección de Valores Extremos, Outliers
 - Los conjuntos de datos que analizamos generalmente proporcionan un subconjunto de datos en el que existe una variabilidad y/o una serie de errores. Estos datos siguen un comportamiento diferente al resto del conjunto ya sea en una o varias variables. Muchas veces es útil estudiarlos para detectar anomalías, mientras que otras veces es mejor descartarlos de los análisis porque ensucian o influyen en los resultados (por ejemplo en los promedios).
 - Orígenes de la variación:
 - Variabilidad de la fuente: Es la que se manifiesta en las observaciones y que se puede considerar como un comportamiento natural de la población en relación a la variable que se estudia
 - Errores del medio: Son los que se originan cuando no se dispone de la técnica adecuada para valorar la variable sobre la población, o cuando no existe un método para realizar dicha valoración de forma exacta. En este tipo de errores se incluyen los redondeos forzados que se han de realizar cuando se trabaja con variables de tipo continuo
 - Errores del experimentador: Son los atribuibles al experimentador, y que fundamentalmente se pueden clasificar de la siguiente forma:
 - Error de Planificación: Se origina cuando el experimentador no delimita correctamente la población, y realiza observaciones que pueden pertenecer a una población distinta
 - Error de Realización: Se comete al llevar a cabo una valoración errónea de los elementos. Aquí se incluyen, entre otros, transcripciones erróneas de los datos, falsas lecturas realizadas sobre los instrumentos de medida, etc.

- Tipos de observaciones:
 - Observación atípica: Es aquel valor que presenta una gran variabilidad de tipo inherente
 - Observación errónea: Es aquel valor que se encuentra afectado de algún tipo de error, sea del medio, del experimentador, o de ambos
- Clasificación—Un proceso de dos pasos
 - Construcción del modelo: Descripción de las clases existentes
 - Cada ejemplo pertenece a una clase determinada
 - El training set es el conjunto de ejemplos que se usa para entrenar el modelo



- Uso del modelo: Para clasificar ejemplos futuros o desconocidos
 - Estimar la precisión del modelo
 - Para esto se aplica el modelo sobre un conjunto de test y se compara el resultado del algoritmo con el real.
 - Precisión es el porcentaje de casos de prueba que son correctamente clasificados por el modelo
 - El conjunto de entrenamiento debe ser independiente del de test para evitar “overfitting”

- Regresión Lineal

- **La regresión lineal aplicada en fabricación es una técnica estadística para modelar e investigar la relación entre dos o más variables.** Este método es aplicable en muchas situaciones en las que se estudia la relación entre dos o más variables o predecir un comportamiento, algunas incluso sin relación con la tecnología. En caso de que no se pueda aplicar un modelo de regresión a un estudio, se dice que no hay correlación entre las variables estudiadas.
- Para poder crear un modelo de regresión lineal, es necesario que se cumpla con los siguientes supuestos
 - La relación entre las variables es lineal
 - Los errores son independientes
 - Los errores tienen varianza constante
 - Los errores tienen una esperanza matemática igual a cero
 - El error total es la suma de todos los errores
- Tipos:
 - Regresión lineal simple: Sólo se maneja una variable independiente
 - Regresión lineal múltiple: Maneja varias variables independientes
- Variables dependientes: Son las variables de respuesta que se observan en el estudio y que podrían estar influidas por los valores de las variables independientes
- Variables independientes: Son las que se toman para establecer agrupaciones en el estudio, clasificando intrínsecamente a los casos del mismo
- Regresión Logística
 - En estadística, la regresión logística es un tipo de análisis de regresión utilizado para predecir el resultado de una variable categórica (una variable que puede adoptar un número limitado de categorías) en función de las variables independientes o predictoras. Es útil para modelar la probabilidad de un evento ocurriendo como función de otros factores. El análisis de regresión logística se enmarca en el conjunto de Modelos Lineales Generalizados (GLM por sus siglas en inglés) que usa como función de enlace la función logit. Las probabilidades que describen el posible resultado de un único ensayo se modelan, como una función de variables explicativas, utilizando una función logística

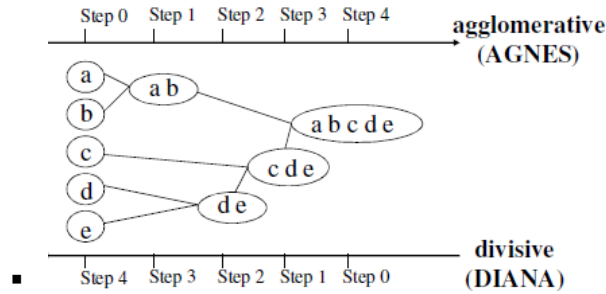
- No Supervisadas

- Clustering

- Cluster: una colección de objetos
 - Similares dentro del cluster
 - Diferentes de los objetos en los otros clusters
- Cluster analisis
 - Agrupar un conjunto de datos en un cluster
- Clustering es clasificación no supervisada: no hay clases predefinidas
- Aplicaciones típicas
 - Como una herramienta independiente para tener una idea sobre la distribución de los datos
 - Como un proceso previo a usar otros algoritmos
- Un buen método de clustering produce clusters de alta calidad con alta similitud dentro de una clase y baja similitud entre las clases. Hay que definir “similitud”
 - Definir la función de distancia es lo importante, y el peso asignado a cada variable
 - **Acá están recortadas varias definiciones de distancia y comparaciones**

- Formas de obtener un cluster:

- Jerárquicas: Usa la matriz de distancia como criterio. No requiere que el número de cluster sea uno de los parámetros de input



- No jerárquicas

- Algoritmo básico:

- Método de particionamiento: Construir una partición de la base de datos D de n objetos en k clusters
 - Dado k encontrar una partición de k clusters que optimice el criterio de partición usado
 - Optimo Global: enumerar todas las particiones posibles
 - Métodos heurísticos: cada cluster está representado por el centro del cluster, o cada cluster está representado por uno de los objetos del cluster

- Jerárquicos Vs No Jerárquicos:

- Agrupamiento Jerárquico

- No hay decisión acerca del número de clusters
 - Existen problemas cuando los datos contienen un alto nivel de error
 - Puede ser muy lento

- Agrupamiento no jerárquico

- Más rápido, más fiable
 - Es necesario especificar el número de clusters (arbitrario)
 - Es necesario establecer la semilla inicial (arbitrario)

- Reglas de Asociación

- Propósito: Generar reglas del tipo: – IF (SI) condición ENTONCES (THEN) resultado
- Tipos de reglas según su utilidad
 - Útiles / aplicables: reglas que contienen buena calidad de información que pueden traducirse en acciones de negocio
 - Triviales: reglas ya conocidas en el negocio por su frecuente ocurrencia
 - Inexplicables: curiosidades arbitrarias sin aplicación práctica
- Cuán buena es una regla? Medidas:
 - Soporte: Cantidad (%) de transacciones en donde se encuentra la regla
 - Ej: “Si B entonces C” está en 4 de 6 transacciones. Soporte (B/C) : 66.6%
 - Confianza: Cantidad (%) de transacciones que contienen la regla referida a la cantidad de transacciones que contienen la cláusula condicional
 - Ej : Para el caso anterior, si B está presente en 5 transacciones (83.33%)
 - Confianza (B/C) = $66.6/83.3 = 80\%$
 - Mejora (Lift (Improvement)): Capacidad predictiva de la regla. $p(B/C) / p(B) * p(C)$
- Tipos de reglas:
 - Booleanas o cuantitativas (de acuerdo a los valores que manejan)
 - Una dimensión o varias dimensiones
 - Con manejo de jerarquías entre los elementos (taxonomías) o con elementos simples

No SQL Databases

- **Definición:** En informática, NoSQL (a veces llamado "no sólo SQL") es una **amplia clase de sistemas de gestión de bases de datos que difieren del modelo clásico del sistema de gestión de bases de datos relacionales (RDBMS)** en aspectos importantes, el más destacado que no usan SQL como el principal lenguaje de consultas. Los datos almacenados no requieren estructuras fijas como tablas, normalmente no soportan operaciones JOIN, ni garantizan completamente ACID (atomicidad, coherencia, aislamiento y durabilidad), y habitualmente escalan bien horizontalmente
 - o Por lo general, los investigadores académicos se refieren a este tipo de bases de datos como almacenamiento estructurado, término que abarca también las bases de datos relacionales clásicas. A menudo, las bases de datos NoSQL se clasifican según su forma de almacenar los datos, y comprenden categorías como clave-valor, las implementaciones de BigTable, bases de datos documentales, y Bases de datos orientadas a grafos
 - o Los sistemas de bases de datos NoSQL crecieron con las principales compañías de Internet, como Google, Amazon, Twitter y Facebook. Estas tenían que enfrentarse a desafíos con el tratamiento de datos que las tradicionales RDBMS no solucionaban. Con el crecimiento de la web en tiempo real existía una necesidad de proporcionar información procesada a partir de grandes volúmenes de datos que tenían unas estructuras horizontales más o menos similares. Estas compañías se dieron cuenta que el rendimiento y sus propiedades de tiempo real eran más importantes que la coherencia, en la que las bases de datos relacionales tradicionales dedicaban una gran cantidad de tiempo de proceso
 - o En ese sentido, a menudo, las bases de datos NoSQL están altamente optimizadas para las operaciones recuperar y agregar, y normalmente no ofrecen mucho más que la funcionalidad de almacenar los registros (p.ej. almacenamiento clave-valor). La pérdida de flexibilidad en tiempo de ejecución, comparado con los sistemas SQL clásicos, se ve compensada por ganancias significativas en escalabilidad y rendimiento cuando se trata con ciertos modelos de datos
- Las características generalmente aceptadas son:
 - o Sin esquema definido
 - o Fácil replicación
 - o Simples APIs
 - o Guardan grandes cantidades de datos
 - o No son ACID (como menciono antes en la definición)
 - o Estos sistemas responden a las necesidades de escalabilidad horizontal que tienen cada vez más empresas
 - o Pueden manejar enormes cantidades de datos
 - o No generan cuellos de botella
 - o Escalamiento sencillo
 - o Diferentes DBs NoSQL para diferentes proyecto
 - o Se ejecutan en clusters de máquinas baratas
- Cómo se guardan los datos
 - o Cassandra guarda los datos en un formato de familia de columnas, cada una con una clave
 - Tiene 2 "interfaces"
 - CLI : tipo UNIX
 - CQL: Tipo SQL
 - o DynamoDB: fue creada y es comercializada por Amazon. Es del tipo clave, valor.
 - El valor puede contener cualquier formato, típicamente especificado en JSON

- Teorema CAP
 - El informática, el teorema CAP, también llamado Teorema de Brewer, establece que es imposible para un sistema de cómputo distribuido garantizar simultáneamente:
 - La consistencia (**Consistency**): Que todos los nodos vean la misma información al mismo tiempo
 - La disponibilidad (**Availability**): La garantía de que cada petición a un nodo reciba una confirmación de si ha sido o no resuelta satisfactoriamente
 - La tolerancia a fallos (**Partition Tolerance**): Que el sistema siga funcionando a pesar de algunas pérdidas arbitrarias de información o fallos parciales del sistema
 - Según el teorema, un sistema puede tener no más de dos de estas tres características simultáneamente.
- Cualquier estrategia de este tipo tiene que tener al menos 3 pasos
 - Detectar la partición
 - Normalmente la detección de la partición está vinculada con un cierto “timeout”
 - Una vez que pasa el tiempo establecido como límite se debe tomar la decisión de si
 - Seguir con la operación afectando la consistencia
 - Cancelar la operación afectando la disponibilidad
 - Entrar en modo “particionado”
 - En esta situación es necesario determinar que operaciones pueden continuar y cuales no
 - Por ejemplo pueden procesarse las altas de clientes y los créditos
 - No pueden procesarse los débitos. En este caso es necesario aclarar a los usuarios que transacciones fueron procesadas, cuales están “en proceso” y cuales deben volver a enviarse
 - Recuperar del modo anterior
 - Cuando la comunicación se restaura es necesario subsanar los problemas de consistencia
 - Gracias al log detallado, el sistema tiene el detalle de lo que ocurrió en cada partición
 - Se tiene que devolver la consistencia y por ahí compensar los errores que se pueden haber producido
 - Esto puede resolverse ordenando ambos logs de una cierta forma y luego ejecutándolos
- Big Table
 - BigTable es un sistema de gestión de base de datos creado por Google con las características de ser: distribuido, de alta eficiencia y propietario. Está construido sobre GFS (Google File System), Chubby Lock Service, y algunos otros servicios y programas de Google, y funciona sobre 'commodity hardware' (sencillos y baratos PCs con procesadores Intel)
 - BigTable almacena la información en tablas multidimensionales cuyas celdas están, en su mayoría, sin utilizar. Además, estas celdas disponen de versiones temporales de sus valores, con lo que se puede hacer un seguimiento de los valores que han tomado históricamente
 - Para poder manejar la información, las tablas se dividen por columnas, y son almacenadas como 'tabletas' de unos 100-200 Mbytes cada una. Cada máquina almacena 100 tabletas, mediante el sistema 'Google File System'. La disposición permite un sistema de balanceo de carga (si una tableta está recibiendo un montón de peticiones, la máquina puede desprenderse del resto de las tabletas o trasladar la tableta en cuestión a otra máquina) y una rápida recomposición del sistema si una máquina 'se cae'
- Map Reduce
 - Es un framework (modelo de programación) utilizado por Google para dar soporte a la computación paralela sobre grandes colecciones de datos en grupos de computadoras y al commodity computing. El nombre del framework está inspirado en los nombres de dos importantes métodos, macros o funciones en programación funcional: Map y Reduce. MapReduce ha sido adoptado mundialmente como una implementación opensource denominada Hadoop, su desarrollo fue liderado inicialmente por Yahoo y actualmente lo realiza el proyecto Apache. En esta década de los años 2010 existen diversas iniciativas similares a Hadoop tanto en la industria como en la academia. Se han escrito implementaciones de bibliotecas de MapReduce en diversos lenguajes de programación como C++, Java y Python.
 - No todos los procesos pueden ser abordados desde el framework MapReduce. Concretamente son abordables sólo aquellos que se pueden disgregar en las operaciones de map() (Map toma uno de estos pares de datos con un tipo en un dominio de datos, y devuelve una lista de pares en un dominio diferente) y de reduce() (La función reduce es aplicada en paralelo para cada grupo, produciendo una colección de valores para cada dominio) y esto es importante a la hora de poder elegir este framework para resolver un problema. Las funciones Map y Reduce están definidas ambas con respecto a datos estructurados en tuplas del tipo (clave, valor).
- Aplicación a cloud computing

DBA

El Administrador de bases de datos (DBA) es el profesional de tecnologías de la información y la comunicación, responsable de los aspectos técnicos, tecnológicos, científicos, inteligencia de negocios y legales de bases de datos

- Implementan, dan soporte y gestionan, bases de datos corporativas
- Crean y configuran bases de datos relacionales
- Son responsables de la integridad de los datos y la disponibilidad
- Diseñan, despliegan y monitorizan servidores de bases de datos
- Diseñan la distribución de los datos y las soluciones de almacenamiento
- Garantizan la seguridad de las bases de datos, incluyendo backups y recuperación de desastres
- Planean e implementan el aprovisionamiento de los datos y aplicaciones
- Diseñan planes de contingencia
- Diseñan y crean las bases de datos corporativas de soluciones avanzadas
- Analizan y reportan datos corporativos que ayuden a la toma de decisiones en la inteligencia de negocios
- Producen diagramas de entidades relacionales y diagramas de flujos de datos, normalización esquemática, localización lógica y física de bases de datos y parámetros de tablas
- Tienen competencias y capacidades en uno o más sistemas de gestión de bases de datos, algunos ejemplos: Microsoft SQL Server, IBM DB2, Oracle MySQL, Oracle database y SQL Anywhere

Incidentes diarios:

- Backups cancelados, Corrupción de datos, Lentitud, Falta de espacio, Errores en general, Incidentes recurrentes, Investigación de comportamientos anómalos, Toda tarea que no tenga una solución definitiva ni conocida

Pedidos diarios:

- Creación de objetos, Movimiento de datos, Duplicación de ambientes, Restores, Pedidos en general

Cambios diarios:

- Instalación de binarios, Creación de bases, Upgrades, Migraciones, Aplicación de parches, Creación de estructuras para nuevas aplicaciones

Problemáticas del puesto:

- Ser soporte o desarrollador, se puede perder confianza de la gente de soporte o perder el contacto con el programador
- Cosas que no conocemos pueden afectar al rendimiento de la base
 - o Aplicaciones cerradas o huérfanas por ejemplo...

Bases de Datos Paralelas y Distribuidas

Definición: Una base de datos distribuida (BDD) es un **conjunto de múltiples bases de datos lógicamente relacionadas las cuales se encuentran distribuidas en diferentes espacios lógicos** (pej. un servidor corriendo 2 máquinas virtuales) e interconectados por una red de comunicaciones. Dichas BDD tienen la capacidad de realizar procesamiento autónomo, esto permite realizar operaciones locales o distribuidas. Un sistema de Bases de Datos Distribuida (SBDD) es un sistema en el cual múltiples sitios de bases de datos están ligados por un sistema de comunicaciones de tal forma que, un usuario en cualquier sitio puede acceder los datos en cualquier parte de la red exactamente como si estos fueran accedidos de forma local. Puede darse el caso de una máquina con muchos procesadores pequeños, o al revés, pequeña cantidad pero poderosos núcleos.

La performance en general se mide de 2 maneras:

- **Throughput:** número de tareas que pueden ser completadas en un intervalo de tiempo
- **Reponse Time:** el tiempo que tarda una tarea en ser completada desde que se solicita

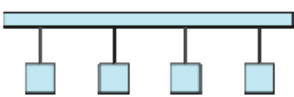
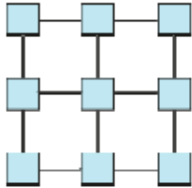
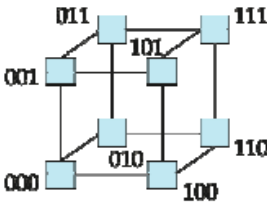
La performance de un sistema distribuido se puede medir:

- **Speed-up:** Más procesadores => Más velocidad; individualmente las queries van más rápido, y se pueden hacer más transacciones por unidad de tiempo
- **Scale-up:** Más procesadores => Más capacidad de procesamiento; la misma query con mayor input tarda el mismo tiempo
 - o **Batch scaleup:** Tamaño del problema
 - o **Transaction scaleup:** Cantidad de problemas

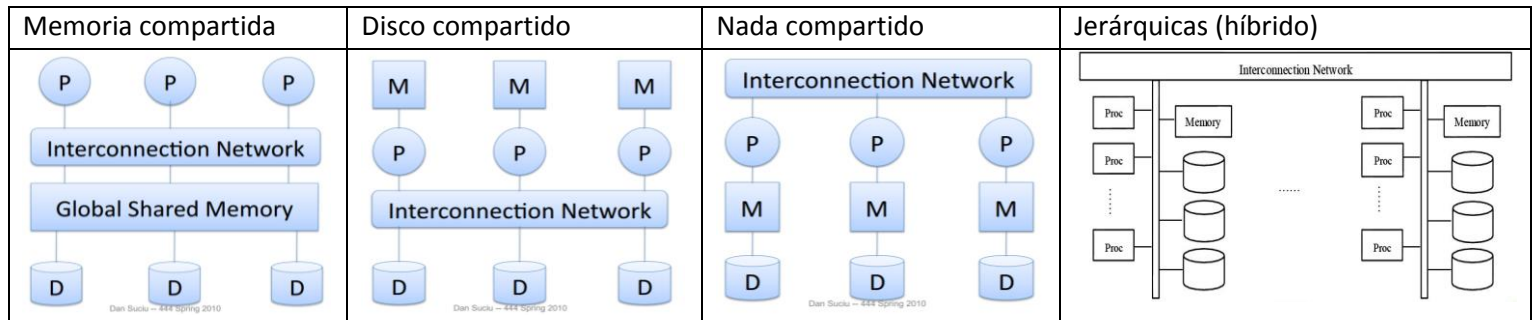
Desde ya que se trata de mantener el crecimiento lineal, es decir: si se aumentan los procesadores aumenta la mejora proporcionalmente. Si aumenta el input, se sigue manteniendo el tiempo (mencionando speed-up y scale-up). Es difícil mantener la linealidad ya que suele predominar el tiempo inicial de la query. La **Interferencia**, es decir: competencia entre varios procesadores por los recursos, suele molestar. También **Skew** (algo así como sesgamiento) es el problema en el que muchas unidades de procesamiento se adaptan al más lento.

Modelos de Interconexión

- Bus: Los componentes se pasan los datos a través de un bus. No escala en paralelismo
- Mesh: Los componentes están como en una grilla, tiene mayor escalabilidad pero es un orden de alrededor raíz de n hacer llegar un mensaje a un componente
- Hypercube: Los componentes se acomodan como en un cubo, cada componente está conectado con log(n) componentes y reduce el delay de un mensaje a log(n)

Bus	Mesh	Hypercube
		

Arquitecturas de las bases paralelas



- **Memoria compartida:**
 - o Ventajas: muy eficiente en cuanto a la comunicación
 - o Desventajas: no escala, porque esa memoria compartida es un cuello de botella
- **Disco compartido:**
 - o Ventajas: no hay cuello de botella, y tiene más tolerancia a fallos
 - o Desventajas: hay otro cuello de botella en la red, a medida que se incrementan los discos
- **Nada compartido:**
 - o Ventajas: no hay interferencia y escala bien
 - o Desventajas: complejidad en cuánto a la comunicación entre los componentes
- **Jerárquicas:**
 - o Características:
 - En el nivel más alto no se comparte nada
 - En un nivel intermedio los componentes se pueden compartir discos
 - En un nivel más bajo los componentes se pueden compartir memorias
 - Pueden mejorar la performance compartiendo memoria virtual

Bases de Datos Paralelas – Detalle

- **Introducción**
 - o Los datos pueden ser distribuidos en varios discos para su procesamiento paralelo
 - o Hay operaciones que pueden distribuirse en varios discos (short, join, etc.)
 - o Las queries en alto nivel se pueden particional más fácil
 - o Diferentes queries pueden correr en paralelo
- **I/O Parallelism**
 - o Reduce el tiempo de adquirir relaciones partiendo el disco, cada relación puede estar en muchos discos
 - o Técnicas:
 - Round Robin: la i -ésima tupla en el disco $i \bmod n$
 - Comparación: El mejor para el scann, pero tarda en encontrar rangos
 - Partición Hash: la función hash sobre un atributo determina el disco
 - Comparación: Bueno para el scann, y para encontrar una tupla, pero sigue fallando en rangos
 - Por Rango: se selecciona un atributo y se distribuyen las tuplas según las particiones del rango
 - Comparación: Bueno para rangos y para el scann, parecería ser el mejor de los 3
 - Hay que tener cuidado del atributo y rango de valores que puede tomar en base a cant discos
 - También hay que tener cuidado con la aglomeración (muchas tuplas en uno y pocas en otro)
 - Muchas técnicas utilizan histogramas para hacer una buena distribución
 - o Evaluación de las técnicas:
 - Tiempo de Scann
 - Tiempo de encontrar una tupla
 - Tiempo de encontrar una tupla en un rango de valores

- Interquery Parallelism

- Las queries/transacciones son ejecutadas en paralelo (por separado)
- Aumenta el Throughput (tran. por seg), no así el Response Time
- Fácil de implementar, especialmente en shared memory, y más difícil en los otros ya que el bloqueo de datos debe hacerse entre varios componentes, y hay que mantener la coherencia de la cache

- Intraquery Parallelism

- Hay 2 formas complementarias de hacer esto:
 - **Intraoperation parallelism:** Paralelizar las operaciones
 - Escala mejor ya que la cant. tuplas por operación < cant. operaciones
 - **Interoperation parallelism:** Ejecutar distintas operaciones de la query
- Los algoritmos asumen: readonly queries, que las arquitecturas no comparten nada, y n cant. discos y proc. Si hay más discos se simula de menos, y que las arq. no comparten nada es compatible con todo
- Intraoperation Parallelism
 - Operations:
 - SORT: Hay un range-partitioning sort, y un parallel external sort-merge, donde este asume que las tuplas ya fueron particionadas entre los discos y c/u ordena su propio contenido
 - JOIN:
 - Join paralelo: Cada procesador computa una parte de las comparaciones
 - Join particionado: Las tuplas se particionan según los rangos
 - Join fragmentado/replicado: Donde no se puede particionar porque hay comparaciones de rango (por ej.), una relación r se particiona y la otra, s, se replica, y todo se paraleliza
 - Hash Join: Asume que s es más chico que r y machea según una función hash la tupla de s a un procesador n. Cada uno obtiene las tuplas que le corresponden a su disco
 - Join anidado: Asuma que s es mucho más chico que r y r se guarda particionado, usa la fragmentación asimétrica replicando s y macheando con r
 - Otras:
 - SELECCIÓN: depende de la distribución de los datos, pero la idea es que un procesador busque si hay condición de igualdad y varios si son de rango o full-scan
 - ELIMINACIÓN, PROYECCIÓN, AGRUPACIÓN, **se deja fuera de este apunte**
 - COSTOS:
 - Descartando varios costos de distribución iniciales, quedaría:
 - $T_{part} + T_{asm} + \max(T_0, \dots, T_n)$, donde:
 - T_{part} : es el tiempo de particionar las relaciones
 - T_{asm} : es el tiempo de reensamblar los resultados
 - T_i : es el tiempo que toma la operación del procesador P_i
 - Tipos de Paralelismo:
 - Pipeline:
 - Si tenemos 4 join, puede mandar a cada núcleo un join y que cada uno vaya devolviendo resultados
 - Sin embargo no escala bien y hay que esperar a que todos devuelvan los resultados
 - Independiente:
 - Se almacenan resultados temporales entre algunos join y luego se combinan, haciendo una especie de pipeline mejorado
 - Optimización: Es más difícil hacerlo teniendo en cuenta las operaciones y tipos de paralelismo
 - Heurísticas:
 - Sin pipelining, solo paralelizar todas las operaciones entre procesadores
 - Elegir el mejor plan secuencial y luego pensar en paralelizar (con o sin pipelining)

- **Design of Parallel Systems**

- Problemáticas normales: Se necesita carga paralela, la probabilidad de la falla de un disco es mayor
- **Distributed Systems**
 - Los datos se propagan a través de las máquinas
 - Las redes interconectan las máquinas, y cada información se comparte entre varias
 - Tipos:
 - Homogéneas: Mismo esquema. Ventaja: permite que sea transparente la distrib. de los datos
 - Heterogénea: Distinto esquema. Ventajas: permite más integración entre sistemas
 - Las queries son más difícil de ejecutarse
 - Transacciones: pueden ser locales o globales
 - TradeOff:
 - Compartir los datos es útil para los distintos sitios
 - La autonomía baja el rendimiento
 - Muchas veces hay redundancia, para bien y para mal por si un nodo se cae
 - Esto trae mucha complejidad para coordinar, aumentando costos y bugs
 - Otra desventaja es poder lograr la atomicidad, se utiliza un protocolo 2PC, u otros
- **Distributed DataBase Systems**
 - Sitios que no comparten nada físico, independientes, y acceden a datos entre si
 - Una base de datos distribuida (BDD) es un conjunto de múltiples bases de datos lógicamente relacionadas las cuales se encuentran distribuidas en diferentes espacios lógicos (pej. un servidor corriendo 2 máquinas virtuales) e interconectados por una red de comunicaciones. Dichas BDD tienen la capacidad de realizar procesamiento autónomo, esto permite realizar operaciones locales o distribuidas. Un sistema de Bases de Datos Distribuida (SBDD) es un sistema en el cual múltiples sitios de bases de datos están ligados por un sistema de comunicaciones de tal forma que, un usuario en cualquier sitio puede acceder los datos en cualquier parte de la red exactamente como si estos fueran accedidos de forma local
 - Ventajas:
 - Transparencia de ubicación, replicación (full o no)
 - Transparencia de fragmentación (vertical u horizontal).
 - Fragmentación vertical: dividir una tupla
 - Ventaja: el acceso a los datos de una tupla es más rápido y permite el paralelismo en el acceso
 - Fragmentación horizontal: distribuir tuplas
 - Ventaja: permite el paralelismo entre varias tuplas, permite agrupar por ciertos criterios
 - Aumento de disponibilidad
 - Mejora la performance y la escalabilidad con el paralelismo
 - Desventajas:
 - Costo de updates
 - Control de concurrencia, solución de compromiso: Elegir una copia primaria donde trabajar

- Nombres:
 - Requisitos:
 - Todos los ítems deben tener un unique name
 - Deben ser fáciles de ubicar
 - Deben poder cambiarse la ubicación de los datos transparentemente
 - Cada sitio debe poder crear nuevos ítems autónomamente
 - Esquemas:
 - Centralizado: Server asigna nombres, cada sitio mantiene guardados datos locales, los sitios le preguntan al server para alojar datos en otro lado, pero esto no satisface el item 4, y puede ser un cuello de botella
 - Uso de Alias: es mejor que el centralizado pero no cumple el item 3
- Transacciones:
 - Cada sitio tiene un transaction manager local y un coordinador para comunicarse con los demás nodos capaz de distribuir tareas
 - Los problemas de red deben ser tenidos en cuenta
 - Commit protocols: 2PC, 3PC (más caro pero corrige algunas cosas del 2PC. No es utilizado)
 - 2PC:
 - Asume fail-stop model: si un sitio se para no hace que los demás también
 - La ejecución la inicia el coordinador
 - Fase 1: Obtener la decisión:
 - El coordinador le pregunta a todos si puede comitear
 - Si uno dice que no, le envía un abort y graba un “no” en el log
 - Si uno dice que si envía y graba un “ready”
 - Fase 2: Grabar la decisión:
 - Si todos dieron el OK el coordinador graba el commit
 - De cualquier forma todos los nodos son avisados de la decisión
 - Problemáticas: en caso de caída asegurar la consistencia viendo los logs de cada uno
 - Blocking problem: ante una caída, todos los sitios tienen que esperar a resolver todos los logs (recovery)
 - Alternativas:
 - Se puede utilizar un sistema de mensajería persistente, que le brinda propiedades transaccionales a los mensajes y trabajar de manera transparente con la red. Hay que tener especial cuidado en diseñar estos sistemas para que no tengan errores
 - Los workflows proveen un modelo transaccional entre sitios

- Control de Concurrencia
 - Single-Lock-Manager Approach:
 - El sistema mantiene un bloqueo unitario de un sitio, cuando una transacción requiere bloquear algo de un sitio se le envía un request al sitio correspondiente y espera el ok.
 - Ventajas: Simple en cuanto a implementación y vista de deadlocks
 - Desventajas: Cuellos de botella, vulnerable a fallas generales
 - Distributed Lock Manager:
 - Hay managers de bloqueos por cada sitio, cada sitio controla el acceso a sus datos
 - Ventajas: Más robusto en cuanto a fallas y mejor distribuido
 - Desventajas: Difícil detectar deadlocks
 - Variantes
 - Primary Copy:
 - Hay una copia principal por item
 - Cuando un sitio solicita el bloqueo se lo solicita al principal y el resto se bloquea luego
 - Ventajas y desventajas del single-lock manager approach
 - Majority Protocol
 - El bloqueo se hace con el consentimiento de la mitad más uno de los nodos que contengan un dato
 - La ventaja es que soporta más las caídas
 - Biased Protocol
 - Permite bloqueos compartidos y exclusivos
 - Da ventajas de funcionamiento para los reads a costa de los writes
 - Quorum Consensus Protocol
 - Una mezcla de los 2 anteriores
 - Timestamping
 - Suponer que cada uno tiene un tiempo y tomarlo como parámetro es sencillo pero tiene el problema de la sincronización de los relojes
 - Para solucionar ese problema se utiliza en reloj lógico con mensajería (tema de SO)
- Replication
 - Problema: + Replicación => + Inconsistencia
 - Posibles enfoques:
 - Master-slave replication: En Master se replica en los Slaves
 - Oracle provee la creación y actualización de snapshots
 - Multimaster and Lazy Replication: Se permiten updates en las réplicas y luego se propaga, aunque aumenta la inconsistencia, ¿cómo contrarrestarla?
 - Centralized Approach: con un coordinador central que construye grafos de dependencias. Puede detectar falsos ciclos y generar falsos rollbacks
 - Disponibilidad y recuperación son más complicados
 - Majority-Based Approach: Se generan bloqueos con más de la mitad de los nodos
 - Muy similar en todo sentido al de las transacciones
 - Comparación con el BackUp remoto
 - Estos últimos están diseñados para dar disponibilidad, son más simples, no necesitan control de concurrencia

- Distributed Query Processing

- En sistemas centralizados lo importante es el acceso a disco, sin embargo en distribuidos se toman en cuenta la cantidad de transmisiones, y ganar performance por paralelismo
- Ya vimos varias posibles optimizaciones, un ejemplo: Select where campo="a" (A U B) se podría separar en select where campo="a" (A) U select where campo="a" (B)
- Una estrategia para el join es el semijoin-strategy que ejecuta operaciones entre sitios proyectando de a partes
- Ejemplo de elección de estrategia:

The result of this query will have 100 tuples, assuming that every department has a manager, the execution strategies are:

1. Transfer Employee and Department to the result site and perform the join at site 3.
 - Total bytes transferred = $1,000,000 + 3500 = 1,003,500$ bytes.
2. Transfer Employee to site 2, execute join at site 2 and send the result to site 3. Query result size = $40 * 100 = 4000$ bytes.
 - Total transfer size = $4000 + 1,000,000 = 1,004,000$ bytes.
3. Transfer Department relation to site 1, execute join at site 1 and send the result to site 3.
 - Total transfer size = $4000 + 3500 = 7500$ bytes.

Preferred strategy: Choose strategy 3.

- Tipos:

- Heterogénea Federada: se menciona como la Homogénea de los sistemas distribuidos
- Heterogénea Multidatabase: se menciona como la Heterogénea de los sistemas distribuidos
- Datos extra:
 - Un sistema multi-base de datos es un software que corre por encima de las bases que da la ilusión que debajo hay una sola base lógica
 - Ventajas: se mantiene la autonomía entre los nodos (por ejemplo cada empresa puede tener su esquema), se puede ver como homogénea desde afuera (tipos, nombres, atributos)
 - Querys: Hay limitaciones de cada base en sus operaciones, cambios de esquema, etc.
 - Directory Systems: es un directorio de información cuya interfaz puede ser web o especializada según un protocolo de acceso. Protocolos:
 - Simplificación del X.500
 - ODBC/JDBC: se produjeron versiones simplificadas en paralelo
 - LDAP (Lightweight Directory Access Protocol)
 - LDAP Data Model: Los directorios guardan entidades con nombres únicos y atributos multivaluados en forma de árbol donde los nodos intermedios son las unidades de la org. y las hojas los objetos específicos
 - Data Manipulation: Se define otro protocolo para DDL y DML, y para el intercambio de archivos donde solo se permite la selección y proyección, con lo cual las querys: deben especificar campos, condiciones y en que base empezar a buscar ya sea por URIs o APIs
 - Ejemplo de URL: `ldap://aura.research.belllabs.com/o=Lucent,c=USA??sub?cn=Korth`
 - Las APIs definen creación y actualización, aunque no soportan atomicidad

Finales Resueltos

Las respuestas a dichos finales son propias del que escribe dicho resumen, por lo que podrían contener errores (aunque esperemos que no)

11/04/2012

Criterio de aprobación: Se aprueba con 12/18.5 puntos, y sin errores conceptuales

- 1) Defina la durabilidad de una transacción y de un ejemplo donde se aplique esta propiedad (1)
 - La durabilidad es la propiedad que asegura que una vez realizada la operación, ésta persistirá y no se podrá deshacer aunque falle el sistema
 - Esta propiedad se aplica en cualquier transacción, por ejemplo, si realizamos un update en un dato y terminamos exitosamente, cualquier otra consulta realizada sobre ese dato dará el resultado que guardamos
- 2) Defina grafo de precedencia y enuncie el teorema de serializabilidad. ¿Por qué es importante este teorema en el control de concurrencia? (2.5)
 - Un grafo de precedencia es un grafo dirigido sobre una historia dada cuyos nodos son las transacciones y cuyos arcos van de una transacción a otra si alguna operación de una transacción precede y conflictúa con alguna operación de otra
 - El teorema de la serializabilidad nos dice que H es serializable sí y solo si su grafo de precedencia es acíclico
 - El teorema es importante ya que si decimos que H es serializable, por definición estamos diciendo que es **equivalente** a una historia serial. Las historias seriales, donde las operaciones de las transacciones están ordenadas, son las que no van a tener problemas de concurrencia, y por lo tanto serán correctas
- 3) Indique y explique dos diferencias entre el lockeo binario y el shared-lock (o lockeo ternario). (2)
 - El bloqueo binario tiene solo dos estados, con la desventaja que si 3 transacciones requieren la lectura de un mismo dato X, no podrán hacerlo al mismo tiempo, mientras que el bloqueo compartido permite dicho comportamiento
 - Otra diferencia a raíz de la anterior se ve en la construcción del grafo de dependencias. Mientras que en el bloqueo binario los arcos se generan en base a los bloqueos, en el bloqueo ternario no se adicionan arcos entre dos bloqueos de lectura. Esto genera una mayor flexibilidad en la cantidad de historias serializables que se encuentran
- 4) Defina la clausura de un atributo. Escriba la definición de clave candidata en función de la clausura. (1.5)
 - La clausura de un atributo es el conjunto de atributos que dependen funcionalmente de un atributo
 - Si $X^+ = R$ (la clausura es toda la relación), entonces X es **superclave** de R. Luego, las claves candidatas son las superclaves tal que no contienen un subconjunto que también sea superclave
- 5) Dada la siguiente relación {NROPEDIDO, CODART, NOMART, CANTPEDIDA, PRECIO, CODEMP., NOMEMP., FECHA}. Indicar :
 - a. Una dependencia funcional completa
 - b. Una dependencia funcional parcial
 - c. Una dependencia funcional transitiva (1.5)
 - Mi idea es que un Nro de Pedido tenga muchos artículos, con lo que la clave principal sería NroPedido y CodArt, el CodEmp que es único y depende del NroPedido
 - Una dependencia funcional completa podría ser: NroPedido, CodArt \rightarrow Precio (precio por unidad, de ese pedido en particular)
 - Una DF parcial se podría ser: NroPedido, CodArt \rightarrow NomArt donde NomArt depende de CodArt
 - Supongamos que el NroPedido \rightarrow CodEmp y que CodEmp \rightarrow NomEmp, luego una dependencia funcional transitiva sería NroPedido \rightarrow NomEmp

- 6) Cuál es la diferencia entre una clave primaria y un índice único? (1)
- Una clave es un conjunto de atributos que definen unívocamente a una relación, y un índice es una estructura que permite acceder a los datos de manera más rápida, el índice único podría ser en base a cualquier clave candidata, mientras que la clave primaria es la elección de una clave candidata que podría o no coincidir
- 7) Indique los tres niveles que tiene la arquitectura de una base de datos. Ejemplifique (1.5)
- Nivel externo: es el más cercano al usuario. En este nivel se describen los datos o parte de los datos que más interesan a los usuarios
 - Por ejemplo, se ven los reportes finales de un usuario de un sistema
 - Nivel conceptual: En este nivel se representan los datos que se van a utilizar sin tener en cuenta aspectos como lo que representamos en el nivel interno
 - Por ejemplo, un diagrama de entidad relación donde tenemos una abstracción de lo que nos importa de la realidad analizada
 - Nivel Interno: es el nivel más cercano al almacenamiento físico de los datos. Permite escribirlos tal y como están almacenados en la base. En este nivel se diseñan los archivos que contienen la información, la ubicación de los mismos y su organización, es decir se crean los archivos de configuración
 - Por ejemplo, serían los stored procedure que tenemos almacenados para su ejecución
- 8) Mencione dos reglas del álgebra relacional que toma en cuenta la optimización por reglas. Ejemplifique su uso. (2)
- Una regla sería llevar las selecciones lo más cercano posible a las hojas del árbol (las relaciones), de manera de lograr la ejecución temprana reduciendo así el número de tuplas que se propagan hacia niveles superiores
 - Otra podría ser descomponer las listas de atributos de las proyecciones y llevarlas lo más cercano posible a las hojas del árbol (relaciones), creando nuevas proyecciones cuando sea posible de manera de no propagar hacia niveles superiores atributos innecesarios. De esta manera se logra una reducción temprana del tamaño de las tuplas, y se reduce la cantidad de bloques necesaria para almacenamiento intermedio
- 9) De un ejemplo de cómo funciona el recovery en el caso de una caída abrupta de la base de datos por un corte de luz si la misma utiliza redo logging? (2)
- Para la respuesta se utilizará la variante sin checkpoints a fin de simplificar el concepto del redo logging
 - Primero dividimos las transacciones en comiteadas y no comiteadas
 - Con las no comiteadas no tenemos problema ya que no bajaron a disco, las abortamos
 - Con las comiteadas tenemos que bajar todo a disco nuevamente, ubicándonos desde el principio del log vamos rehaciendo las operaciones de las transacciones ya comiteadas
- 10) Detalle el uso que el DBMS le da al system catalog en el momento de crear una tabla (1)
- Cuando creamos una tabla o un índices lo registramos en el system catalog, este se actualiza periódicamente con información estadística de los datos que contiene y permite estimar la selectividad de los operadores
 - Podemos consultarlo como a cualquier tabla, pero obviamente no podemos modificarlo (sería muy riesgoso)
- 11) ¿Por qué es necesario para un DBA conocer la cantidad de registros que inicialmente va a tener una tabla? (1)
- Para responder esta pregunta, nos tenemos que enfocar en las responsabilidades de un DBA
 - Ellos diseñan la distribución de los datos y las soluciones de almacenamiento, entonces: ¿cómo podrían diseñar la distribución si no saben con cuántos registros contarán?
 - Por otro lado crean y configuran las BD, con los índices inclusive, por lo que conocer la cantidad de registros es importante. No es lo mismo considerar tener un índice para una tabla de 20 registros que plantear la creación de un índice para una tabla de millones, teniendo en cuenta como se van a buscar los datos
 - Con respecto a los backups, también tienen que planificar el espacio y el tiempo que van a llevar
 - También se puede estimar el tamaño del archivo físico que va a contener los datos y configurar en cuanto se agranda el archivo cuando se llena. Si va a tener 100 registros no tiene sentido hacer un alloc de 1 GB y si sabemos que va a crecer mucho no tiene sentido hacer un archivo de 4 MB y que cuando se reubica en disco pida de a 1 MB por vez

12) ¿Cuándo una entidad se considera débil? De un ejemplo (1.5)

- Una entidad es débil cuando necesita a otra para identificarse, ya que sus atributos identificatorios son parte de la otra entidad. La participación de esta entidad con la otra, siempre es total ya que no puede existir una entidad débil sin su respectivo “padre”

Fines del 2012

Las preguntas son aproximadas, y no están todas.

- ¿Qué propiedad de las trans. es la que permite que una transacción vuelva atrás si hubo una falla? (de las ACID)
 - Atomicidad ya que o se ejecuta todo, o no se ejecuta nada
- Enumere las relaciones entre conjuntos que permite el AR
 - Unión, Intersección, Resta, Producto Cartesiano, División, y los Joins
- De un ejemplo de una interrelación binaria (DER)
 - Por ejemplo: Profesor – Materia (es de muchos a muchos)
- ¿Qué es un Datawarehouse y qué utilidad tiene?
 - Es una colección de datos orientada a un determinado ámbito
 - Debe ser orientado a temas (elementos relativos al mismo evento deben estar agrupados), variables en el tiempo, no volátil (la información no se modifica) e integrado (debe tener los datos de la organización y deben ser consistentes). Su actualización es periódica, y los datos provenientes de distintos sistemas operacionales, con esquemas posiblemente distintos. Estos datos pasan por distintas etapas para llegar a integrar los datos finales
 - Utilidades:
 - Ayuda a la toma de decisiones en la entidad en la que se utiliza
 - Es una base de datos diseñada para favorecer el análisis y la divulgación eficiente de datos
 - Es un expediente completo de una organización
 - No se utiliza para el uso transaccional, y tampoco se piensa para soportar updates/inserts/deletes
 - Por lo general se encuentra desnormalizado
- ¿Quién es el encargado de mantener el System Catalog?
 - El DBSM es el único capaz de actualizar el catálogo, ya que este contiene información estadística de los atributos de las tablas (entre otras cosas) y permitir alguna modificación por parte de un usuario es altamente riesgoso
- Si tenés un sistema con undo-logging, hacer un gráfico para mostrar en qué caso se hace una recuperación
 - No se entiende bien la pregunta, ya que la recuperación se hace siempre hay una falla producida por un problema en el suministro eléctrico, los discos, u otros. Lo del gráfico se debe referir a hacer un log y mostrar como se recupera un determinado caso. Basta con mostrar un ejemplo del apunte:
 - Caso en rojo: Acontece un crash y el último registro del log es: <T,B,8> (Paso 4) La única transacción involucrada es T. Como no encontramos el registro commit para T, la clasificamos como no comiteada. Luego, todas sus acciones deben deshacerse, desde el paso 4 (el último anterior al crash) hasta el principio del log. Luego, obtenemos las siguientes acciones al aplicar el mecanismo Undo.
 - Caso en Rosa: Acontece un crash y el último registro del log es: <Commit T> (Paso 5) Como T es una transacción comiteada, ignoramos todas sus acciones. En este caso, es la única transacción, por lo que no se efectúan cambios en la recuperación.
- Te dan dos transacciones y piden una historia serial y una serializable
 - Acá lo que habría que hacer es hacer la historia serial, y modificar una operación que no sea conflictiva de tal forma que queden dos historias equivalentes, una serial y otra serializable. Otra posible forma sería hacer un grafo de dependencias de una historia serial y con el algoritmo que está en el apunte sacar las historias equivalentes

# Paso	Registro
1	<Start T>
2	<T,A,8>
3	<T,A,16>
4	<T,B,8>
5	<Commit T>

30/07/2013

Las preguntas son aproximadas, y no están todas, pero están bastante completas

2) Defina detalladamente el problema de la falsa sumarización

- Se refiere a cuando uno hace una operación donde agrupa varios registros (ejemplo una suma de un atributo) y en el medio se produce un update en dichos datos, el valor resultante no es ni la del estado anterior ni siguiente al update

3) Defina equivalencia de dos transacciones

- Lo que creo es que en realidad lo que se busca es la equivalencia entre historias
- Dos historias (H1 y H2) son equivalentes si:
 - o Están definidas sobre el mismo conjunto de transacciones
 - o Las operaciones conflictivas tienen el mismo orden (dos operaciones de T_i y T_j ($i < j$)) son conflictivas si operan sobre el mismo ítem y al menos alguna de las dos es un write)

4) Dada la siguiente relación {NROPEDIDO; CODART, NOMART, CANTPEDIDA, PRECIO, CODEMP. NOMEMP., FECHA}. Indicar:

a) Una dependencia funcional completa; b) Una dependencia funcional parcial; c) Una dependencia funcional transitiva. (1.5)

- [Contestada en el final del 11/04/2012]

5) ¿Qué es un Plan de Ejecución?

- Es un árbol de ejecución que define cómo se resolverá una consulta dada, indicando paso a paso los algoritmos y estructuras que se usarán para resolverla

6) Defina superclave y cual es la diferencia con la clave candidata

- Una superclave es un conjunto de atributos tal que su clausura funcional es equivalente al total de la relación
- Una clave candidata es, en cambio una superclave tal que no existe un subconjunto que también sea superclave

7) 2 restricciones que modelen cosas del mundo real que haya en el MER y como se implementan en la base

- Una restricción en el mundo podría ser cada televisor tiene una marca, dicha relación generará una interrelación entre las entidades marca y televisor, donde televisor tendrá participación total con respecto a la marca
 - o Posible implementación: Serían dos tablas: Televisor y Marca, y el Televisor tendría un campo Marca_Id (por ej.) con una constraint de foreign key a Marca
- Otra restricción podría ser la siguiente: un estudiante podría ser de dos tipos: o de grado o de posgrado. Dicha restricción en el modelo se podría ver como una jerarquía de estudiantes (cada uno con sus atributos particulares)
 - o Posible implementación: Asumiendo que es disjoint tendríamos la tabla Estudiante, EstudianteGrado, EstudiantePosgrado donde EstudianteGrado y EstudiantePosgrado tiene un campo con FK a Estudiante y en Estudiante se guarda también un campo con el discriminante

8) Defina lo que es un select en algebra relacional y de 2 propiedades

- Un select en AR es un filtro que permite seleccionar un subconjunto de tuplas de una relación (R), todas aquellas que cumplan con dicha condición
- Dos propiedades importantes utilizadas en la optimización son:
 - o Cascada: Es equivalente una selección con muchas condiciones a un a muchas selecciones cada una con partes de las condiciones anteriores
 - o Conmutatividad: Es indistinto en que orden estén declaradas las selecciones

9) No sé que cosa con la clausura with grant (o algo similar)

- El Grant concede permisos sobre un elemento protegible a una entidad de seguridad.
- El concepto general es GRANT <algún permiso> ON <algún objeto> TO <algún usuario, inicio de sesión o grupo>
- La pregunta apunta a WITH GRANT, que es una cláusula opcional en el GRANT propiamente dicho, que además le concede al usuario afectado permisos para ejecutar el comando GRANT sobre ese permiso particular que se le asigna
 - o Un ejemplo sería: GRANT EXECUTE ON TestProc TO TesterRole WITH GRANT OPTION

10) Como se recupera la BD con undo/redo

- El objetivo es deshacer las transacciones no comiteadas y rehacer las comiteadas, esto es, combinar los mecanismos de Undo y Redo. El mecanismo es:
 - o 1ero deshacer las transacciones no comiteadas desde el último registro hasta el principio
 - o 2do rehacer las transacciones comiteadas desde el principio hasta el final
 - o 3ero se agrega un registro abort para cada transacción no comiteada, y se hace un flush del log

11) Instrucción para actualizar el catalog

- Por un lado existe en Oracle la posibilidad de forzar un UPGRADE del catálogo y simplemente es UPGRADE CATALOG
- La instrucción en ANSI SQL es UPDATE STATISTICS y fuerza la actualización más allá del período automático definido
- Sin embargo no creo que la pregunta apunte a conocer eso, sino a decir lo siguiente:
 - o Las actualizaciones del catálogo por parte del usuario no se permiten, son responsabilidad del BDMS. Si permitiéramos actualizar el catálogo a los usuarios de la BD sería altamente riesgoso

12) 2 operaciones que realice el DW

- Creo que la pregunta no debió ser así en el final, hagamos 2 interpretaciones
 - o La que tuve inicialmente fue que se busca saber 2 operaciones de un ETL (Extract, Transform, Load)
 - Las operaciones son Migración, Limpieza, Transformación, Carga, y Conciliación (Validación)
 - Veamos los primeros dos
 - Migración: El propósito de la migración es mover los datos de los sistemas operacionales a las áreas de trabajo (staging areas), evitando mover datos innecesarios
 - Limpieza: La idea acá es corregir, estandarizar y completar los datos, identificar redundancias, valores atípicos o perdidos, uniformar las denominaciones de las demás bases y estandarizar atributos entre otros
 - o Otra podría ser apuntando a las operaciones OLAP, las cuales son (mencionando todas y no solo dos):
 - ROLL UP (Te alejás en la dimensión)
 - DRILL DOWN (Te enfocás en algo de la dimensión)
 - SLICE (Recortar una rebanada del cubo en base a una dimensión)
 - DICE (recortar de a varias dimensiones armando subcubos)

13) ¿Por qué el DBA tiene que conocer el crecimiento de la base?

- Para responder esta pregunta, veamos las operaciones que tiene que realizar un DBA
 - o Crean y configuran bases de datos relacionales: Es importante tener en cuenta la envergadura de la base para tener en cuenta la creación de índices entre otras cosas
 - o Diseñan la distribución de los datos y las soluciones de almacenamiento: Es importante tener en cuenta que si la base crece demasiado una solución distribuida podría ser una mejor opción a una centralizada
 - o Garantizan la seguridad de las bases de datos, incluyendo backups y recuperación de desastres: Las estrategias de los backups no son las mismas si hablamos de una base chica con poco crecimiento a una grande
 - o Otra sería también para poder calcular el espacio en disco que se va a necesitar
 - o Podría haber más, habría que ver la sección DBA de este apunte